On Efficient 3D Object Retrieval

Hao Liu
The Hong Kong University of Science and Technology
hliubs@cse.ust.hk

Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
raywong@cse.ust.hk

***Abstract*—Due to the growth of the 3D technology, digital 3D models represented in the form of point clouds have attracted a lot of attention from both industry and academia. Due to a variety of applications, we study a fundamental problem called the *3D object retrieval*, which is to find a set of 3D point clouds stored in a database that are similar to a given query 3D point cloud. To the best of our knowledge, solving the problem of 3D object retrieval *efficiently* remains unexplored in the research community. In this paper, we propose a framework called C_2O to find the answer efficiently with the help of an index built on the database. In most of our experiments, C_2O performs up to 2 orders of magnitude faster than all adapted algorithms in the literature. In particular, when the database size scales up to 100 million points, C_2O answers the 3D object retrieval within 10 seconds but all adapted exact algorithms need more than 1000 seconds.**

***Index Terms*—Point cloud, 3D object retrieval**

I. Introduction

Recently, digital 3D models in the form of point clouds are gaining popularity in both industry and academia due to the increasing use of low-cost 3D depth sensors and LiDAR sensors [1], [2]. In industry, both hardware and software are using 3D point clouds. For hardware, some recent mobile devices like iPhone 14 Pro and Samsung's Galaxy S22 use LiDAR or other technology to obtain 3D point clouds easily. For software, Google is using 3D point clouds in their products, such as generating 3D street view images from LiDAR point clouds in Google Maps [3]. 3D point clouds have also been applied in various industrial domains, such as machine construction [4], civil engineering [5] and medical image analysis [6]. In academia, there are a lot of research projects on point clouds (e.g., 3D reconstruction [7], autonomous driving [8]) and VR/AR [9]). Thus, studying problems on point clouds could be interesting in the database community (especially, the spatial database community). In Figure 1, we show five 3D objects each of which is represented by a *point cloud* where a point cloud is defined to be a set of points in a 3D space.

In most (if not all) of the applications on 3D point clouds, *3D object retrieval* is one fundamental problem. Specifically, we are given a set $\mathcal{P}$ of a number of point clouds (each representing a 3D object). Given a query point cloud Q and a user parameter δ where δ is a non-negative real number, we want to find a set of point clouds in $\mathcal{P}$ such that for each point cloud P in the set, Q is *similar* to a portion of P. More formally, the *distance* between Q and a portion of P is at most δ. It is worth mentioning that under our problem setting, Q could be similar to the entire P (which is a special case of a (full) portion of P). If δ is set to 0, it is an exact match between Q and a portion of P. If δ is set to a value larger than 0, it is a similarity search problem between Q and a portion of P. Consider Figure 1 where $\mathcal{P}$ contains Objects 1–4. Suppose that our query point cloud is shown in the figure and δ is set to a positive value near to 0. Our 3D object retrieval returns Object 1 and Object 2 in the answer (not Object 3 and Object 4) since both Object 1 and Object 2 contain the back rest part which is similar to the query but both Object 3 and Object 4 do not.

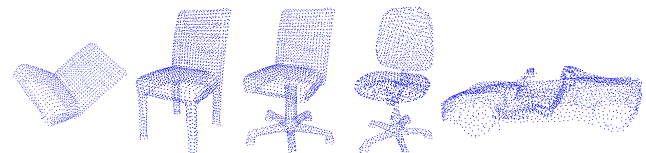


Fig. 1. A Motivating Example

3D object retrieval has various applications. (1) *3D Object Database Search:* In the machine construction industry, due to the growth of 3D printing, many companies provide a database $\mathcal{P}$ containing all 3D models each representing a machine component for other companies to search for a particular 3D model Q [4]. (2) *Unmanned Vehicle Localization:* When a 3D indoor scene is constructed in the form of a 3D point cloud (in database $\mathcal{P}$), an unmanned vehicle could determine its location by using its LiDAR sensors to obtain a point cloud as a query point cloud Q which is used to find which portion of the whole 3D indoor scene point cloud [10]. This is vital in either indoor or a territory outside the Earth when GPS signals are inaccurate or unavailable [11]. (3) *Scientific Database Search:* In a scientific database $\mathcal{P}$ like a celestial database containing all stars in the galaxy, scientists would like to find a list of regions which have star patterns similar to a given star pattern Q [12].

To the best of our knowledge, there are two branches of existing studies closely related to our problem. The first branch is called *registration* [13], [14], [15], [16], [17], [18], which, given two point clouds P_1 and P_2 with similar numbers of points, is to determine whether P_1 could be translated or rotated such that the *whole* resulting point cloud is roughly equal to P_2 (known as *full registration* [13], [14], [15], [16]), or a *large portion* of P_1 (e.g., more than 60% as required in existing studies) is roughly equal to a *large portion* of P_2 (known as *partial registration* [17], [18]). Note that the registration problem (either full registration or partial registration) is different from our 3D object retrieval problem. Firstly, it only considers "mapping" *two* given 3D point clouds but we consider how to "search" for a given query point cloud in a database involving *many* 3D point clouds. Secondly, partial registration considers "mapping" two given 3D point clouds with *large* portions but we consider "mapping" two 3D point clouds (one from the query 3D point cloud and the other from the database) in any arbitrary portion (which could be large or small).

The second branch is called *similarity search* (or called *shape matching*) [19], [20], [21], [22], which is to find a set of *different* portions of a given database point cloud representing a 3D scene such that each portion "looks" similar to a query object Q (e.g., the back rest part of a chair) in the point cloud form. Specifically, each portion of a given database point cloud, as well as Q, is encoded as an *embedding vector* by the same machine learning model. These studies about similarity search are to return a set of portions whose embedding vectors have their Euclidean distance to the query embedding vector at most a given threshold value. However, since "embedding vectors" could only capture the shape of 3D objects *roughly* not *exactly*, the 3D point clouds returned in the output may not really be a real chair, and thus, the accuracy is not high. For instance, when the query object is the back rest part of a chair (as shown in Query of Figure 1), a car (as shown in Object 4 of Figure 1) is unfortunately returned by [19] in our experiment (because the shape of the car is *wrongly* captured into an embedding vector similar to the embedding vector of the query object), but the expected outputs are chairs with similar back rest parts (e.g., Object 1–3 of Figure 1). It is worth mentioning that some recent studies [19], [20] in the similarity search branch name their problem as "object retrieval", which is still different from *our* "object retrieval" problem definition and has the same inaccuracy issue. Moreover, [19], [20] do not consider rotation/translation and thus give incorrect results easily. For example, [19], [20] fail to return the expected chairs (e.g., Object 1–3 of Figure 1) for a query with the *rotated* back rest part of a chair (as shown in Query of Figure 1).

Unfortunately, none of the adapted and existing algorithms could solve our 3D object retrieval problem well. Firstly, although no existing algorithms, originally designed for registration, directly solve the 3D object retrieval problem, we adapt these existing algorithms, namely Super4PCS [23] and GoICP [24], for our problem. All of these adapted algorithms suffer from one of the following problems. The first problem is low efficiency, particularly when the database size is large. The second problem is a very bulky index size (for those adapted algorithms using indices). Secondly, existing algorithms originally designed for similarity search, namely PointNetVLAD [19] and PCAN [20], are not accurate as described previously.

Motivated by this, we propose a novel and concise representation called *donut* to denote some important "features" of a 3D point cloud so that whenever we want to compare two point clouds P and Q, instead of comparing the coordinates of all the points in P with the coordinates of all the points in Q, we just need to compare the donut representation of P with the donut representation of Q. Since the donut representation of a point cloud is concise, it is easier to do a comparison. Specifically, the donut representation has the following advantages: *translation-invariant* and *rotation-invariant*. When a point cloud is compared with another point cloud, since these 2 point clouds are in 2 different spaces, we have to perform a translation operation and a rotation operation (which are very time-consuming). The donut representation of a point cloud is said to be *translation-invariant* (*rotation-invariant*) if no matter whether we perform a translation (rotation) operation on the point cloud, the donut representation does not change. It is important that the donut representation is translation-invariant and rotation-invariant because it could avoid the time-consuming operations of translation and rotation. When both the query point cloud Q and all database point clouds are encoded by the donut representation, it is more efficient to find a list of database point clouds which are similar to Q.

In this paper, we propose an algorithm called C_2O which involves 2 phases, namely the preprocessing phase and the query phase. In the preprocessing phase, C_2O first builds an index on all database point clouds based on the donut representation. In the query phase, given a query point cloud Q, C_2O generates a *small* set of candidates denoting the possible "features" from Q by our novel pruning strategies, and finds a set of point clouds in the database efficiently based on this small set with the help of the index.

Our contributions are as follows. Firstly, we study the *efficient* 3D object retrieval problem, which, to the best of our knowledge, remains unexplored since efficiency is the main focus in the database community and has not been studied extensively in the literature. Secondly, we propose a novel and concise representation called donut to effectively capture the important features of 3D point clouds so that retrieving similar 3D objects is very effective and efficient due to its translation-invariant property and its rotation-invariant property. Thirdly, based on the donut representation, we propose an index-based algorithm called C_2O to find a set of similar 3D objects efficiently. Fourthly, we conducted comprehensive experiments to show that our proposed algorithm C_2O is very effective and efficient. In particular, our proposed algorithm is 1-3 orders of magnitude faster than the adapted algorithms. We scale the database size up to 100 million points, a large data volume that has been applied in various real-world applications (e.g. localization [18], object recognition [25], and 3D reconstruction [26]), and our proposed algorithm took less than 10 seconds for retrieving 3D objects but all adapted exact algorithms took more than 1000 seconds, which is not acceptable in real-life applications. Besides, the f-measure of our proposed algorithm is 100% but the f-measure of the existing algorithms originally designed for similarity search is around or less than 15% (with both precision and recall less than 25%), which is not acceptable too.

In the rest of this paper, we give the formal definition of our 3D object retrieval in Section II. In Section III, we first describe our framework C_2O. Then, we present the detail of our algorithm C_2O in Section IV. In Section V, we introduce the relevancy of existing problems and existing algorithms. In Section VI, we present our experimental results. Finally, we conclude our paper in Section VII.

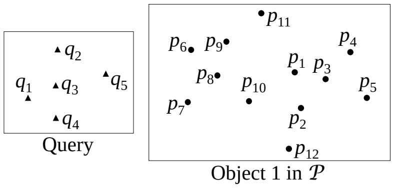


Fig. 2. Examples of Database and Query

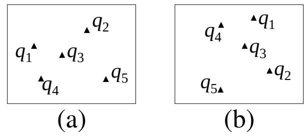


Fig. 3. Examples of Rotated Query Point Clouds: (a) Rotated with 45° Clockwise, (b) Rotated with 135° Clockwise

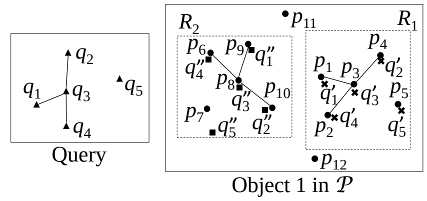


Fig. 4. Examples of Transformation and Retrieval

II. Problem Definition

Consider a 3-dimensional space. A *point cloud* is defined to be a set of 3-dimensional points where each point is represented in a 3-dimensional xyz coordinate. The size of a point cloud is defined to be the total number of points in this point cloud. For example, in Figure 1, all the points in "Query" form a point cloud and all the points in Object 1 form another point cloud. We are given a database $\mathcal{P}$ which contains $|\mathcal{P}|$ point clouds. Each point cloud in $\mathcal{P}$ is associated with an ID. We define the size of a database, denoted by n, to be the sum of the sizes of all point clouds in the database. Let us create a toy example as shown in Figure 2 where we have one query point cloud and we have the database $\mathcal{P}$ containing many objects but we just show Object 1 in $\mathcal{P}$. Here, we illustrate the example in a 2-dimensional space (for easier illustration) but the same illustration can be applied to a 3-dimensional space.

To compare two point clouds in two different coordinate systems, we have to perform a *rigid transformation* on one point cloud so that the transformed point cloud is in a consistent coordinate system with the other point cloud. A rigid transformation Θ on a point cloud is formed by a rotation operation and a translation operation. In geometry, a rotation operation is denoted by a 3D rotation 3×3 matrix R and a translation is denoted by a translation 3-dimensional vector t. A rigid transformation Θ is represented by a 2-tuple (R, t). Given a point p (represented in a 3-dimensional vector) and a transformation $\Theta = (R, t)$, the *transformed point* of p wrt Θ, denoted by $T_\Theta(p)$, is defined as follows: $T_\Theta(p) = Rp + t$. That is, point p is rotated with R and is translated with t. Given a point cloud P, we define the *transformed point cloud* of P wrt Θ, denoted by $T_\Theta(P)$, to be $\{T_\Theta(p) \mid p \in P\}$. Figures 3(a) and 3(b) show the query point cloud from Figure 2 rotated with 45^o clockwise and 135^o clockwise, respectively.

In our problem, we need to compare a query point cloud with a database point cloud. We just need to do a rigid transformation on one point cloud, which is chosen as the query point cloud. Figure 4 shows the rotated query point clouds in Figures 3(a) and 3(b) in the coordinate system of Object 1, each undergoing a translation operation, as indicated in dashed rectangles R_1 and R_2, respectively.

For easier illustration, we first define the distance between two point clouds in the same coordinate system, after which we consider the case when the two point clouds are in different coordinate systems. Consider two point clouds in the same coordinate system, namely Q and P. For each point q in Q, we define the *correspondence point* of q on P, denoted by $corr(q, P)$, to be the point in P that has the smallest Euclidean distance to q (following a common definition [27]). We denote the Euclidean distance between point q and point p by $\|q, p\|$. The distance between Q and P in the same coordinate system, denoted by $dist_{same}(Q, P)$, is defined as follows based on the concept of the *average* L_2-norm.

$$dist_{same}(Q, P) = [\tfrac{1}{|Q|} \textstyle\sum_{q \in Q} \|q, corr(q, P)\|^2]^{\frac{1}{2}} \tag{1}$$

Note that the above distance definition has been widely applied in [24], [27], [13], [23]. While other distance definitions (e.g., the *exact distance* [28], [29] and the *Hausdorff distance* [30]) exist in the literature, they have obvious issues. For instance, the exact distance (which returns 0 if Q and P are exactly equal, and 1 otherwise) can only tell whether Q and P are the same, but cannot measure how similar they are, and the Hausdorff distance (which returns the maximum $corr(q, P)$ for any $q \in Q$) is easily affected by an outlier in Q far away from its correspondence point on P. Our applied distance definition avoids the above issues by using the average L_2-norm, which could effectively measure the similarity of two point clouds even when noise and outliers are present [13].

Consider back Figure 4. We know that the rotated query point cloud Q' in R_1 is in the same coordinate system of Object 1. Let P be the point cloud denoting Object 1. In this figure, we know that the correspondence point of q'_1 (q'_2) on P is p_1 (p_4). That is, $corr(q'_1, P) = p_1$ and $corr(q'_2, P) = p_4$. After we find the correspondence point of each point in Q', we can compute the distance between Q' and P (i.e., $dist_{same}(Q', P)$). Since *visually*, each query point in Q' is very close to a point in P, the distance between Q' and P is very small (e.g., near to 0). It is easy to have a similar conclusion for the rotated query point cloud Q'' in R_2. Note that since q''_5 is a little bit far away from its closest point in P (i.e., p_7), we derive that $dist_{same}(Q'', P) > dist_{same}(Q', P)$.

However, as mentioned before, typically, two given point clouds are usually not in the same coordinate system. We have to perform a rigid transformation on one point cloud (in our case, the query point cloud). Consider two point clouds in different coordinate systems, namely Q and P. Given a transformation Θ on Q, the distance between Q and P in different coordinate systems wrt Θ, denoted by $dist_{diff}(Q, P|\Theta)$, is defined as follows.

$$dist_{diff}(Q, P|\Theta) = [\tfrac{1}{|Q|} \textstyle\sum_{q' \in T_\Theta(Q)} \|q', corr(q', P)\|^2]^{\frac{1}{2}} \tag{2}$$

It is the same as Equation 1 except that we include the rigid transformation Θ on Q only in the above equation.

Let Θ_o be the optimal rigid transformation in a set $\mathcal{T}$ of all possible transformations of Q such that Equation 2 is minimized. We overload symbol $dist_{diff}(\cdot, \cdot)$ and define the distance between Q and P in different coordinate systems, denoted by $dist_{diff}(Q, P)$, to be the following.

$$dist_{diff}(Q, P) = [\frac{1}{|Q|} \sum_{q' \in T_{\Theta_o}(Q)} \|q', corr(q', P)\|^2]^{\frac{1}{2}} \quad (3)$$

In the following, for the sake of simplicity, when we write $dist(Q, P)$, we mean $dist_{diff}(Q, P)$.

Definition II.1 (3D Object Retrieval)**.** Given a query point cloud Q and a user parameter δ where δ is a non-negative real number, we want to find a set of point clouds in $\mathcal{P}$ such that for each point cloud P in the set, $dist(Q, P) \leq \delta$.

III. FRAMEWORK C_2O

In our 3D object retrieval problem, we have to check whether a given query point cloud Q and a given point cloud P in the database $\mathcal{P}$ have $dist(Q, P) \leq \delta$. We propose our framework called C_2O which involves the following 3 major steps to complete this goal.

- **Step 1 (Coarse Transformation):** We first perform a transformation Θ on Q based on *some* "representative" points of Q so that the transformed Q and P are in the same coordinate system. Since the transformation Θ is based on some "representative" points of Q (not *all* points of Q), we call Θ a *rough* or *coarse* transformation. The reason to have a coarse transformation is that finding an *optimal* transformation requires a time-consuming search that considers all possible transformations involving all points [24]. In a typical experimental setting (e.g., with the database size 1M and the query size 100), finding an optimal transformation from *scratch* takes over 1000 seconds but finding a rough transformation from some points in Q takes about 0.5 second only.
- **Step 2 (Complete Transformation):** We then perform a *complete* transformation Θ_o on Q based on *all* points of Q according to the initial coarse transformation Θ so that the transformation on Q is optimal. With the help of Θ, finding the optimal transformation is much faster. In the typical experimental setting described above, it takes 0.9 second only.
- **Step 3 (Object Retrieval):** With the optimal transformation Θ_o on Q, we can compute $dist_{diff}(Q, P|\Theta_o)$ ($= dist(Q, P)$) easily and check whether $dist(Q, P) \leq \delta$. If yes, P will be in the answer of our 3D object retrieval.

The most challenging step in framework C_2O is coarse transformation, because finding a "close-to-optimal" transformation based on some "representative" points of Q could be costly (though much cheaper than finding the optimal one) since searching "representative" points blindly may not help to improve the efficiency.

One may ask how to obtain some "representative" points of Q in Step 1 so that we could execute coarse transformation efficiently? In this paper, we propose a novel and concise representation of a given point cloud called the *donut* representation. Specifically, each point in a given point cloud can be encoded in a representation called the *relative-distance representation* which is represented in the form of a 6-dimensional tuple where each dimension in this tuple is a real number. For each point $p_{(1)}$ in P, we find 3 other points in P in a particular order, say $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, based on the principle of the *regular tetrahedron* (to be elaborated in detail later) and compute the relative-distance representation according to all the 6 pairwise distances among these 4 points. The relative-distance representation of point $p_{(1)}$ wrt the other 3 points (i.e., $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$), denoted by $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$, is defined as follows. That is, $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)}) =$

$$\begin{aligned}&(\|p_{(1)}, p_{(2)}\|, \|p_{(1)}, p_{(3)}\|, \|p_{(1)}, p_{(4)}\|,\\ &\|p_{(2)}, p_{(3)}\|, \|p_{(2)}, p_{(4)}\|, \|p_{(3)}, p_{(4)}\|)\end{aligned} \quad (4)$$

We define the *first, second, third, fourth owners* of this representation to be $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, respectively. The donut representation of a point cloud is defined to be a collection of relative-distance representations of all points in this point cloud. More importantly, since the relative-distance representation considers only *relative* distance computation, as described in Section I, this donut representation (or the relative-distance representation) satisfies the translation-invariant property and the rotation-invariant property. These properties ensure that given a set S of 4 points, even if we obtain a set S' of 4 points by arbitrarily rotating/translating the 4 points in S where each of these points may deviate from its original coordinate with some distance, the relative-distance representation generated based on S is "close" to the one generated based on S'.

With the concept of "relative-distance representation", one may give the following naive implementation of framework C_2O when we consider a query point cloud Q and *one* data point cloud P. In Step 1, we check the *relative-distance representation* of a point q in Q (says R_q) and the *relative-distance representation* of each point p in P (says R_p). If R_q and R_p are close, we can find the (coarse) transformation Θ according to R_q and R_p. In Step 2, with Θ, we derive a (complete) transformation Θ_o using the state-of-the-art algorithm called Go-ICP [24] guaranteed to return the optimal transformation. In Step 3, we output P if $dist_{diff}(Q, P|\Theta_o) \leq \delta$. Since typically, there is *more than one* point cloud in the database, the naive implementation has to process each database point cloud one-by-one, which is time-consuming.

In this paper, we propose an index-based approach which involves 2 phases, namely the preprocessing phase and the query phase. In the preprocessing phase, we build an index on all point clouds in $\mathcal{P}$ according to the concept of "relative-distance representation". In the query phase, given a query point cloud, we select one point in Q and use its relative-distance representation to find the similar representations stored in the index efficiently.

IV. ALGORITHM C_2O

We first give the details of the donut representation (containing a number of relative-distance representations of points in a point cloud) in Section IV-A. Then, we describe how the relative-distance representation of a point in a *data* point cloud could be matched with that of a point in a *query* point cloud in Section IV-B. Finally, we give the details of our proposed algorithm C_2O in Section IV-C.

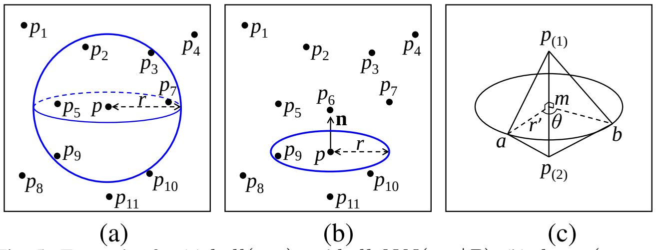


Fig. 5. Examples for (a) $ball(p, r)$ and $ball\text{-}NN(p, r|P)$, (b) $donut(p, r, \mathbf{n})$ and $donut\text{-}NN(p, r, \mathbf{n}|P)$, and (c) Regular Tetrahedron Formation

A. Donut Representation

In this section, we give the details of the donut representation. The major principle of this donut representation is to generate a 6-dimensional tuple (called the *relative-distance representation*) according to the concept of "regular tetrahedron". We use this concept for two reasons. The first reason is due to the usage of a tetrahedron involving 4 points, since using 4 points to represent some local structures of a point cloud could give a more differentiating power for pruning [13] (compared with using 3 points [31]). The second reason is due to the usage of "regularity", since more regular shapes could have more differentiating power [32]. Note that our proposed concept of "regular tetrahedron" is different from existing 4-point representations [13], [32], [33]. Firstly, existing representations of 4 points are separated into two point-pairs and a connector connecting the two point-pairs, which leads to the costly two-step pruning that is first based on the two point-pairs and then based on the connector. We consider a "holistic" shape of 4 points (i.e., a tetrahedron), and thus our pruning can be efficiently done in one step. Secondly, the 4 points in our representation are *ordered* but the existing 4-point representations are *unordered*, so we avoid enumerating all 4-point combinations, required by existing representations, to match the query 4-point representation in the database. Thus, our representation is much more efficient.

Given 4 points in a 3D space, namely $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, a *tetrahedron* is defined to be a polyhedron composed of 4 triangular faces, 6 straight edges and 4 vertex corners where $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$ form the 4 corners. We also say that this tetrahedron is represented by these 4 points. The *size* of a tetrahedron is defined to be the maximum length of an edge in the tetrahedron. A tetrahedron is said to be *regular* if the lengths of edges are equal. Given a non-negative real number r, a tetrahedron is said to be an *r-sized regular tetrahedron* if the size of this tetrahedron is r and this tetrahedron is regular.

Consider a regular tetrahedron represented by $p_{(1)}, p_{(2)}, p_{(3)}$ and $p_{(4)}$. Let m be the mid-point between $p_{(1)}$ and $p_{(2)}$. It can be observed that $\beta = \angle p_{(3)} m p_{(4)}$ is the *dihedral angle* between the two adjacent faces of the edge formed by $p_{(1)}$ and $p_{(2)}$ (i.e., face $p_{(1)}p_{(2)}p_{(3)}$ and face $p_{(1)}p_{(2)}p_{(4)}$). It is easy to verify that β is equal to $\arccos 1/3$ radian, and the length of both line $p_{(3)}m$ and line $p_{(4)}m$ is equal to $(\sqrt{3}/2)\|p_{(1)}, p_{(2)}\|$.

We are ready to describe the relative-distance representation. Specifically, given a data point cloud P, for each point $p_{(1)}$ in P, we construct its relative-distance representation as follows.

- **Step 1 (Forming Tetrahedron):** We find 3 other points in P from $p_{(1)}$ in a *particular* order, namely $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, such that the 4 points found form a tetrahedron (represented by these 4 points) that is as close as an R-sized regular tetrahedron where R is a non-negative real number roughly equal to a user parameter r, a non-negative real number.
- **Step 2 (Constructing Relative-Distance Representation):** According to these 4 points found, we construct the relative-distance representation (which is a 6-dimensional tuple) based on all the pairwise distances among these 4 points.

Step 2 is straightforward. Next, we give the details of Step 1. We are given a point $p_{(1)}$ in a point cloud P. Due to the unbalanced distribution of points in a point cloud, it is nearly impossible to construct an exact regular tetrahedron from $p_{(1)}$ and 3 other points in P. Thus, in the following, we describe how we find the 3 other points in P in a particular order such that the constructed tetrahedron is "close" to a regular tetrahedron. We first define the concepts of "ball" and "donut".

Given a 3D point p and a non-negative real number r, the *r-sized ball of p*, denoted by $ball(p, r)$, is defined to be the surface of a sphere centered at p with radius equal to r, and the *nearest neighbor of the r-sized ball of p* in P, denoted by $ball\text{-}NN(p, r|P)$, is defined to be the point in P which is the nearest to a point in the r-sized ball of p (i.e., $ball(p, r)$) and has its Euclidean distance to p at least r. Figure 5(a) shows an example of $ball(p, r)$ (marked in blue) and that point p_{10} in P is $ball\text{-}NN(p, r|P)$.

Next, we define the concept of "donut". Given a 3D point p, a non-negative real number r and a 3D vector $\mathbf{n}$, the *r-sized donut of p with its normal* $\mathbf{n}$, denoted by $donut(p, r, \mathbf{n})$, is defined to be the boundary of a circle centered at p with radius equal to r which is on the plane with its (surface) normal equal to $\mathbf{n}$, and the *nearest neighbor of the r-sized donut of p with its normal* $\mathbf{n}$ in P, denoted by $donut\text{-}NN(p, r, \mathbf{n}|P)$, is defined to be the point in P which is nearest to a point in the r-sized donut of p with its normal $\mathbf{n}$ (i.e., $donut(p, r, \mathbf{n})$). Figure 5(b) shows an example of $donut(p, r, \mathbf{n})$ (marked in blue) and that point p_9 in P is $donut\text{-}NN(p, r, \mathbf{n}|P)$.

Now, based on the concepts of "ball" and "donut", we can give our principle of constructing a tetrahedron T involving 4 points from P (i.e., $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$) where T is not necessarily regular in most cases. To achieve that, we first construct a *virtual* regular tetrahedron involving 4 points in a particular order, where the first 2 points are in P (which are assigned to $p_{(1)}$ and $p_{(2)}$, respectively), and the latter 2 points are *virtual* points, namely a and b (which may not be in P). The size of this regular tetrahedron (denoted by R) is determined in the following process but its value is near to a user parameter r (note that r can be easily chosen experimentally as we introduce in Section VI). In the final tetrahedron T involving 4 points (which come from P), the first 2 points are exactly $p_{(1)}$ and $p_{(2)}$, and the final 2 points, namely $p_{(3)}$ and $p_{(4)}$, are "close" to points a and b, respectively.

The steps of finding these 4 points in P (together with points a and b) are described as follows. We start with a point in P, say $p_{(1)}$. According to a user parameter r (which is a non-negative real number), we find another point in P, say $p_{(2)}$, which is the nearest to $p_{(1)}$ with distance from $p_{(1)}$ at least r (i.e., $ball\text{-}NN(p_{(1)}, r|P)$). Based on $p_{(1)}$ and $p_{(2)}$, we form a virtual regular tetrahedron as follows where the size of this tetrahedron (i.e., R) is equal to $\|p_{(1)}, p_{(2)}\|$ (typically, R is close to r in practice and in our experiments, R is at most 1.1 times r). We construct a *locus* of a point such that this point has its distance to $p_{(1)}$ and its distance to $p_{(2)}$ both equal to $\|p_{(1)}, p_{(2)}\|$. Let m be the mid-point between $p_{(1)}$ and $p_{(2)}$, and $\mathbf{n}$ be a vector from $p_{(2)}$ to $p_{(1)}$. It is easy to verify that this locus is equal to $donut(m, r', \mathbf{n})$, where $r' = (\sqrt{3}/2)\|p_{(1)}, p_{(2)}\|$. Since the shape of the locus is similar to "donut", we thus call this as the donut representation. According to this locus, we find the point in P nearest to this locus (i.e., $donut\text{-}NN(m, r', \mathbf{n}|P)$), which is assigned to $p_{(3)}$. Then, the point on the locus nearest to $p_{(3)}$ can be determined and is assigned to a. Based on $p_{(1)}, p_{(2)}$ and a, we can virtually construct a regular tetrahedron with 4 vertices, whose first 3 vertices are $p_{(1)}$, $p_{(2)}$ and a. Clearly, the last vertex of this virtual regular tetrahedron (i.e., b) can be found on two possible positions along the locus. For a more deterministic formation, we choose the position of point b (among these 2 positions) to be the one closest to a in the anti-clockwise direction from a on the locus (in the view point from $p_{(1)}$ to $p_{(2)}$). Figure 5(c) illustrates point b in a regular tetrahedron. Then, we find the point in P nearest to b, which is assigned to $p_{(4)}$. Finally, we obtain all 4 points in P (i.e., $p_{(1)}, p_{(2)}, p_{(3)}$ and $p_{(4)}$).

Consider our example as shown in Figure 6(a). Suppose we want to form a tetrahedron starting from point p_6 (i.e., $p_{(1)} = p_6$). We first assign p_{11} as $p_{(2)}$, since $p_{11} = ball\text{-}NN(p_6, r|P)$. Secondly, let m be the mid-point between p_6 and p_{11}. Let $\mathbf{n}$ be a vector from p_{11} to p_6. Let $r' = (\sqrt{3}/2)\|p_{(1)}, p_{(2)}\|$. We denote $donut(m, r', \mathbf{n})$ by D. p_9 is assigned to $p_{(3)}$ since $p_9 = donut\text{-}NN(m, r', \mathbf{n}|P)$. Note that a is the nearest point to p_9 on donut D. Thirdly, in the figure, b is a point on D which is in the anti-clockwise direction from a (indicated by the blue arrow) (in the view point from p_6 to p_{11}). We find the nearest point in P to b (i.e., p_{10}) which is assigned to $p_{(4)}$.

Note that the above steps generate a tetrahedron (represented by $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$) close to a regular tetrahedron (represented by $p_{(1)}$, $p_{(2)}$, a and b). With $p_{(1)}, p_{(2)}, p_{(3)}$ and $p_{(4)}$, we can construct the relative-distance representation as shown in Equation 4.

B. Matching Relative-Distance Representation

Now, we describe how the relative-distance representation of a point in a *data* point cloud can be matched with that of a point in a *query* point cloud even though there is an order enforced when we find the 3 other points from a point (e.g., $p_{(1)}$).

Consider a query point cloud Q and a data point cloud P in $\mathcal{P}$. In our problem, we want to determine whether $dist(Q, P) \leq \delta$. When δ is greater than 0, this means that after an optimal transformation on Q resulting in a transformed point cloud Q', each point q' in Q' has its correspondence point p on P such that $\|q', p\|$ *could* be greater than 0. However, since δ is a fixed value, $\|q', p\|$ can be upper bounded by a fixed value too. The following lemma shows the upper bound of $\|q', p\|$ according to δ.

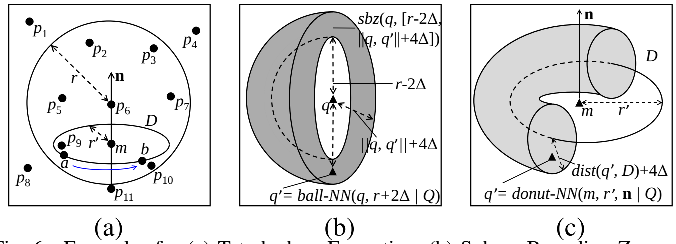


Fig. 6. Examples for (a) Tetrahedron Formation, (b) Sphere Bounding Zone and (c) Donut Bounding Zone

Lemma IV.1. *Let Q' be the point cloud optimally transformed from Q (wrt P). If $dist(Q, P) \leq \delta$, then for each point q' in Q' and its correspondence point p on P, $\|q', p\| \leq \delta|Q|^{1/2}$.*

Proof sketch. $\|q', p\|^2 \leq \sum_{q' \in T_{\Theta_o}(Q)} \|q', corr(q', P)\|^2 \leq \delta^2|Q|$ by Equation 3. Thus, $\|q', p\| \leq \delta|Q|^{1/2}$. □

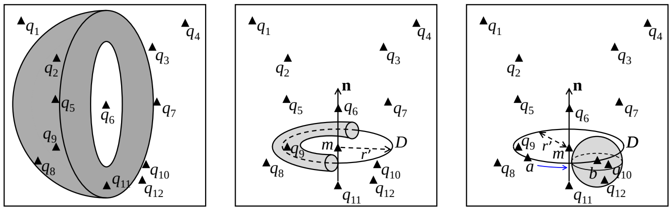
Let $\triangle = \delta|Q|^{1/2}$. Based on this lemma, we derive a property called *Distance Bound Property* stating that the distance between each (transformed) query point and its correspondence on P is bounded by $\triangle$. Note that the bound $\triangle$ is tight since it is possible that $\|q', p\| = \triangle$.

Now, we describe how the relative-distance representation of a point in the query point cloud Q (generated based on 4 points in Q) can be used to "map" with the relative-distance representation of a point in the data point cloud P (generated based on 4 points in P). This involves the following steps.

- **Step (a) (Constructing Relative-Distance Representation Candidates):** We construct a set of *candidates* of the relative-distance representation of a point in Q, say $q_{(1)}$.
- **Step (b) (Finding Answers from Database):** For each *candidate* c in Step (a), we find a list of point clouds in $\mathcal{P}$ according to the relative-distance representations of all point clouds in $\mathcal{P}$ and the candidate c.

We first assume that $q_{(1)}$ is an arbitrarily selected point in Q. Later, in Section IV-C2, we introduce our strategy to find the point $q_{(1)}$ in Q which could lead to the optimal efficiency. The core problem is how to construct a set of candidates of the relative-distance representation of $q_{(1)}$ *efficiently* in Step (a). Consider a point cloud P in $\mathcal{P}$ where $dist(Q, P) \leq \delta$. Since each point in P has its relative-distance representation wrt 3 other points in P in a *particular order*, one method for effective mapping between Q and P based on their relative-distance representations is described as follows. Let $p_{(1)}$ be the correspondence point on P of $q_{(1)}$ (i.e., an arbitrary point in Q) when we compute $dist(Q, P)$. We should find an ordering of 3 other points in Q, say $q_{(2)}, q_{(3)}$ and $q_{(4)}$, whose correspondence points on P are the second, third and fourth owners of the relative-distance representation of $p_{(1)}$ in this *order*.

One naive but *inefficient* implementation of Step (a) is to enumerate *all combinations* in $\{q_{(1)}\} \times Q \times Q \times Q$, denoting all possible orderings of 4 points in Q starting from $q_{(1)}$. Instead, we would like to do it efficiently by pruning many combinations and finding a *small* (candidate) set of these



(a) Finding $S_{(2)}$ (b) Finding $S_{(3)}$ (c) Finding $S_{(3)}$

Fig. 7. Examples of Finding $S_{(2)}$ / $S_{(3)}$ / $S_{(4)}$

combinations only, based on two major principles. The first principle is to follow Distance Bound Property and the second principle is to follow how to find the 4 points in a point cloud (as described in Section IV-A). Once we obtain the candidate set in Step (a), we can do Step (b) easily by comparing each candidate in Step (a) with point clouds in $\mathcal{P}$ according to their relative-distance representations. Later, in Section IV-C, we describe how we use an index to further speed up this step.

Now, we propose a method to find a set of candidates much more *efficiently* for Step (a). First, starting from point $q_{(1)}$, we find sets $S_{(2)}$ (Section IV-B1), $S_{(3)}$ (Section IV-B2) and $S_{(4)}$ (Section IV-B3) of candidates points in Q for points $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$, respectively, based on the two major principles. Then, we construct a candidate set of all relative-distance representations according to $\{q_{(1)}\}$, $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ (Section IV-B4). This candidate set corresponds to the output of Step (a). Note that $S_{(i)}$ is much smaller than Q for each $i \in [2, 4]$, and thus this candidate set is much smaller than the set of all combinations (i.e., $\{q_{(1)}\} \times Q \times Q \times Q$).

1) Finding $S_{(2)}$: We describe how to find $S_{(2)}$ according to $q_{(1)}$ based on the two principles. Before that, we first define some concepts whose major ideas come from the two principles. Given two non-negative real numbers, namely r_1 and r_2, where $r_1 \leq r_2$, we define an *interval*, denoted by $[r_1, r_2]$, to represent all values such that each value is at least r_1 and at most r_2. Given a 3D point p and an interval $[r_1, r_2]$ where $r_1 < r_2$, the *sphere boundary zone of p with interval* $[r_1, r_2]$, denoted by $sbz(p, [r_1, r_2])$, is defined to be the 3D space containing all 3D points such that each point in this space has its distance to p at least r_1 and at most r_2.

In Figure 6(b), the shaded region is $sbz(q, [r_1, r_2])$ where $r_1 = r - 2\triangle$, $r_2 = \|q, q'\| + 4\triangle$ and $q' = ball\text{-}NN(q, r + 2\triangle|Q)$. For better illustration, in the figure, we show $sbz(q, [r_1, r_2])$ and similar concepts in the "half-sphere", and the other "half-sphere" is just symmetric. Note that it is obvious $r_1 = r - 2\triangle \leq r \leq r_2 = \|q, q'\| + 4\triangle$.

Next, we show our idea of constructing $S_{(2)}$ that contains the desired $q_{(2)}$ having $p_{(2)}$ as its correspondence point on P. Since $p_{(2)} = ball\text{-}NN(p_{(1)}, r|P)$ (and is thus close to $ball(p_{(1)}, r)$ in most cases), based on the two principles, we ensure that $q_{(2)}$ is also close to $ball(q_{(1)}, r)$ or $\|q_{(2)}, q_{(1)}\|$ is close to $\|p_{(2)}, p_{(1)}\|$. We can thus find a set $S_{(2)}$ of *candidate* points in Q for $q_{(2)}$ such that each candidate point in $S_{(2)}$ is inside a sphere bounding zone near $ball(q_{(1)}, r)$. Specifically, we define the procedure of finding $S_{(2)}$ with $q_{(1)}$ in Q, denoted by $find\text{-}S_{(2)}(q_{(1)}|Q)$, as follows. We first find a point q' to be the nearest neighbor of a $(r + 2\triangle)$-sized ball of $q_{(1)}$ in Q. Then, we find a set $S_{(2)}$ of *candidate* points in Q for $q_{(2)}$ inside $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$ such that $S_{(2)}$ must contain our desired $q_{(2)}$ and can be listed out. The details are presented in our technical report [34].

Figure 7(a) shows an example of finding $S_{(2)}$ in Q if we pick q_6 as $q_{(1)}$. The shaded region (shown in the "half-sphere") represents $sbz(q_6, [r - 2\triangle, \|q_6, q'\| + 4\triangle])$ where q_{11} is selected to be q' since $q_{11} = ball\text{-}NN(q_6, r + 2\triangle|Q)$. Thus, $S_{(2)}$ is the set of all the points inside the shaded region (i.e., $S_{(2)} = \{q_2, q_8, q_9, q_{11}\}$). Note that q_1, q_3, q_4, q_5, q_6, q_7, q_{10} and q_{12} in the figure are outside the shaded region.

2) Finding $S_{(3)}$: Next, we consider how to construct $S_{(3)}$ according to $q_{(1)}$ and $q_{(2)}$ based on the two principles where $q_{(2)} \in S_{(2)}$. Before that, we also define some concepts. Given a point q' and a donut D, the distance between q' and D, denoted by $dist(q', D)$, is defined to be the distance between q' and its nearest point on D. Given a donut D and an interval $[r_1, r_2]$, we define the *donut boundary zone of D with an interval* $[r_1, r_2]$, denoted by $dbz(D, [r_1, r_2])$, to be the 3D space containing all 3D points such that each point in this space has distance to donut D at least r_1 and at most r_2. In Figure 6(c), the shaded region is $dbz(D, [0, dist(q', D) + 4\triangle])$.

Now, we construct $S_{(3)}$ similarly given $q_{(1)}$ and $q_{(2)}$ (whose correspondence points on P are $p_{(1)}$ and $p_{(2)}$, respectively). In Section IV-A, for a *database* point cloud P, the third owner of the relative-distance representation of $p_{(1)}$ (i.e., $p_{(3)}$) is the nearest point in P to a donut (constructed from $p_{(1)}$ and $p_{(2)}$). Thus, the desired $q_{(3)}$ with $p_{(3)}$ as its correspondence point on P is also close to a donut constructed similarly from $q_{(1)}$ and $q_{(2)}$. Specifically, we define the procedure of finding $S_{(3)}$ with $q_{(1)}$ and $q_{(2)}$ in Q, denoted by $find\text{-}S_{(3)}(q_{(1)}, q_{(2)}|Q)$, as follows. We find the donut D to be $donut(m, r', \mathbf{n})$, where m is the mid-point between $q_{(1)}$ and $q_{(2)}$, $r' = (\sqrt{3}/2)\|q_{(1)}, q_{(2)}\|$ and $\mathbf{n}$ is a vector from $q_{(2)}$ to $q_{(1)}$. Then, we find a set $S_{(3)}$ of *candidate* points in Q for $q_{(3)}$ inside region $dbz(D, [0, dist(q', D) + 4\triangle])$, where $q' = donut\text{-}NN(m, r', \mathbf{n}|Q)$, such that $S_{(3)}$ must contain our desired $q_{(3)}$ and can be listed out. The details are also presented in [34].

Consider the example shown in Figure 7(b) where we pick q_6 as $q_{(1)}$ and q_{11} as $q_{(2)}$. Accordingly, we find $D = donut(m, r', \mathbf{n})$ where m is the mid-point between q_6 and q_{11}, $r' = (\sqrt{3}/2)\|q_6, q_{11}\|$ and $\mathbf{n}$ is the vector from q_{11} to q_6. The shaded region represents $dbz(D, [0, dist(q', D) + 4\triangle])$ where q_9 is selected to be q' since $q_9 = donut\text{-}NN(m, r', \mathbf{n}|Q)$. Thus, $S_{(3)} = \{q_9, q_{10}\}$ since they are inside the shaded region.

3) Finding $S_{(4)}$: We also consider how to construct $S_{(4)}$ according to $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ (whose correspondence points on P are $p_{(1)}$, $p_{(2)}$ and $p_{(3)}$, respectively) where $q_{(2)} \in S_{(2)}$ and $q_{(3)} \in S_{(3)}$. Since the fourth owner of the relative-distance representation of $p_{(1)}$ (i.e., $p_{(4)}$) is close to the second virtual point in the steps in Section IV-A, we can find the desired $q_{(4)}$ with $p_{(4)}$ as its correspondence point on P near the second virtual point in Q constructed in the same way. Specifically, we define the procedure of finding $S_{(4)}$ with $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ in Q, denoted by $find\text{-}S_{(4)}(q_{(1)}, q_{(2)}, q_{(3)}|Q)$, as follows. We find point a and b following the same steps in Section IV-A.

Then, we find point q'' to be the nearest neighbor of point b in Q. Finally, we find set $S_{(4)}$ containing all points in Q inside $sbz(b, [0, \|b, q''\| + 4\triangle])$, such that $S_{(4)}$ must contain our desired $q_{(4)}$ and can be listed out. See [34] for details.

In the example shown in Figure 7(c), we pick q_6 as $q_{(1)}$, q_{11} as $q_{(2)}$ and q_9 as $q_{(3)}$. Point a is the nearest point of q_9 on D, and point b is a point on D which is in the anti-clockwise direction from a (indicated by the blue arrow) (in the view point from q_6 to q_{11}). We find q_{10} to be the nearest neighbor of b in Q (i.e., $q'' = q_{10}$), and thus we find $sbz(b, [0, \|b, q''\| + 4\triangle])$ shown as the shaded region in the figure. Since the shaded region also covers q_{12}, $S_{(4)} = \{q_{10}, q_{12}\}$.

4) Constructing Candidate Set: Now, we present how to construct the candidate set of relative-distance representations according to $\{q_{(1)}\}$, $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ (i.e., the details of Step (a)). Let $\mathcal{C}$ be a set of relative-distance representations to store the results of the candidates, initialized to be an empty set.

- **Step (i) (Finding $S_{(2)}$):** We find $S_{(2)} = find\text{-}S_{(2)}(q_{(1)}|Q)$. For each point $q_{(2)}$ in $S_{(2)}$, we do the following step.
  - **Step (i)(1) (Finding $S_{(3)}$):** We find $S_{(3)} = find\text{-}S_{(3)}(q_{(1)}, q_{(2)}|Q)$. For each point $q_{(3)}$ in $S_{(3)}$, we do the following step.
    - **Step (i)(1)(I) (Finding $S_{(4)}$):** We find $S_{(4)} = find\text{-}S_{(4)}(q_{(1)}, q_{(2)}, q_{(3)}|Q)$. For each point $q_{(4)}$ in $S_{(4)}$, we do the following step.
      - **Step (i)(1)(I)-1 (Constructing all candidates):** Firstly, we obtain the relative-distance representation $R = rd(q_{(1)} \ |q_{(2)}, q_{(3)}, q_{(4)})$ by Equation 4. Secondly, we associate R with $q_{(1)}$, $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$. Thirdly, we insert R into the result set $\mathcal{C}$ of candidate relative-distance representations.

The following lemma shows the correctness of the above details for Step (a). Specifically, if $dist(Q, P) \leq \delta$, the output candidate set generated by Step (a) contains a relative-distance representation of an arbitrary point $q_{(1)}$ in Q, which has a *correspondence* relative-distance representation of a point on P, says $p_{(1)}$. For the sake of space, a proof sketch of each lemma/theorem is given in this paper but the full proof can be found in our technical report [34].

Lemma IV.2. *Consider a query point cloud Q and a database point cloud P. Let $q_{(1)}$ be a point in Q where $corr(q_{(1)}, P)$ is $p_{(1)}$. Let $p_{(2)}, p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of $p_{(1)}$, respectively. Let $\mathcal{C}_{q_{(1)}}$ be the output set of candidate representations of $q_{(1)}$ on Q. If $dist(Q, P) \leq \delta$, then there exist three points $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$ in Q, such that $p_{(i)} = corr(q_{(i)}, P)$ for $i \in [2, 4]$, and $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) \in \mathcal{C}_{q_{(1)}}$.*

Proof sketch. Since $dist(Q, P) \leq \delta$, by Lemma IV.1, for $i \in [2, 4]$, $p_{(i)}$ exists in P, and $\|q_{(i)}, p_{(i)}\| \leq \triangle$. $S_{(i)}$ for $i \in [2, 4]$ must contain $q_{(i)}$ by our steps of finding $S_{(i)}$. Clearly, $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)})$ will be included in $\mathcal{C}_{q_{(1)}}$. □

Note that the candidate set size (i.e., $|\mathcal{C}|$) could be $O(|Q|^3)$, since each of $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ could be $O(|Q|)$ in the worst case. However, our effective pruning strategies to find $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ ensure that $|\mathcal{C}|$ is generally small. In our experiments of typical settings (i.e., $|Q| = 100$), $|\mathcal{C}|$ is only around 400, significantly smaller than $|Q|^3 = 10^6$. In Section IV-C2, we give our strategy to further keep $|\mathcal{C}|$ as small as possible.

C. Two Phases of C_2O

We present the preprocessing phase and the query phase of C_2O in Section IV-C1 and Section IV-C2, respectively.

1) Preprocessing Phase: The preprocessing phase is straightforward with our donut representation. For each point cloud P in $\mathcal{P}$, we do the following sub-steps. The first sub-step is to build a 3D index I_P (e.g., R*-tree[1] [35]) on all points in P. The second sub-step is to perform the following procedure for each point $p_{(1)}$ in P. We construct the relative-distance representation of $p_{(1)}$ (i.e., $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ where $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$ are the other 3 points found (as described before) with the help of index I_P. We insert $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ into a 6-dimensional index I_{DB} (e.g., R*-tree), initialized to an empty index. Note that in I_{DB}, each $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ is associated with two parts: (1) the ID of point cloud P in $\mathcal{P}$ and (2) the point IDs of $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$.

Clearly, the final index size of I_{DB} is $O(n)$ where n is the sum of the sizes of all database point clouds. Next, we discuss the preprocessing time. Firstly, building an index I_P of one point cloud P takes $O(|P| \log |P|)$ time, and the total time complexity of building indices of all point clouds in the database $\mathcal{P}$ is $O(\sum_{P \in \mathcal{P}} |P| \log |P|) = O((\sum_{P \in \mathcal{P}} |P|) \log n) = O(n \log n)$, since each $|P|$ is at most n. Secondly, for each point in a database point cloud P, we obtain the relative-distance representation in expected $O(\log |P|)$ time with index I_P. Since there are n $(\geq |P|)$ database points, obtaining all relative-distance representations takes $O(n \log n)$ time. The overall preprocessing time complexity is $O(n \log n)$.

Note that C_2O is generic for any other multi-dimensional index to implement I_{DB}. In our experiments, we tested different index types, and R*-tree gives the best query efficiency.

2) Query Phase: We describe the following 2 steps of our query phase as follows. Consider a query point cloud Q.

Step 1 (Relative-Distance Representation Candidate Generation): Since the number of the generated candidates is crucial, to keep this number as small as possible, we use a simple and effective strategy as follows. Firstly, for each point in Q, we perform the steps described in Section IV-B and obtain a set of candidate relative-distance representations of this point. Secondly, we select the point in Q such that its candidate set has the smallest size among all the generated candidate sets, and assign this point to $q_{(1)}$. Let $C_{q_{(1)}}$ denote the set of candidate relative-distance representations of $q_{(1)}$.

Step 2 (Point Cloud Matching): We first initialize a variable $\mathcal{R}$, which is to store a set of IDs of the point clouds in P as the query result, to an empty set. Then, we do the following

[1] R*-tree is a tree-like data structure to index points in a multi-dimensional space using minimum bounding boxes (MBB). It is very efficient to perform window queries or nearest neighbor queries using the R*-tree index by pruning many branches (represented by MBBs) that cannot contain any query results.

steps. *Step a:* We initialize a variable $\mathcal{C}$, which is to store a set of 2-tuples each in the form of (R_Q, R_P) where R_Q (R_P) is a relative-distance representation of a point in Q (P), to an empty set. *Step b:* For each representation R_Q in $C_{q_{(1)}}$, we perform a window query on I_{DB} such that for each dimension in the 6-D space, the lower and upper boundary of the query window is $v - 2\triangle$ and $v + 2\triangle$, respectively, where v is the value of R_Q for this dimension, and obtain a result X which is a set of relative-distance representations in I_{DB} inside the query window. Note that each representation R_P in X is associated with an ID i and the point IDs of the 4 owners of R_P. Then, for each R_P in X, we insert a 2-tuple (R_Q, R_P) into $\mathcal{C}$. *Step c:* For each (R_Q, R_P) in $\mathcal{C}$, we perform the following step. Firstly, let i be the ID of R_P. Secondly, if i could be found in $\mathcal{R}$, we do nothing (since there is no need to process R_P because the ID of its point cloud P is in the result $\mathcal{R}$). Otherwise (i.e., if i could not be found in $\mathcal{R}$), we do the following. We perform a coarse transformation Θ based on (1) the 4 owners of R_Q and (2) the 4 owners of R_P. Based on Θ, we perform a complete transformation Θ_o based on (1) all points of Q and (2) all points of point cloud P with ID i. If $dist_{diff}(Q, P|\Theta_o) \leq \delta$, we insert the ID i into $\mathcal{R}$.

The following theorem shows the correctness of C_2O.

Theorem IV.1 (Correctness)**.** *We are given a query point cloud Q and a non-negative real number δ. Let $\mathcal{R}$ be the set of database point clouds returned by our original query phase. Let $\mathcal{R}^*$ be the expected solution of our 3D object retrieval problem. Then, $\mathcal{R} = \mathcal{R}^*$.*

Proof sketch. We need to show that $\mathcal{R}$ contains no false positive or false negative. The former is obvious since we only output P such that $dist(Q, P) \leq \delta$. For the latter, for any $P \in \mathcal{R}^*$, there exists a 2-tuple (R_Q, R_P) in $\mathcal{C}$ such that R_P and P has the same ID (by Lemma IV.2), and thus we will finally perform a complete transformation Θ_o (guaranteed by Go-ICP [24] to be the optimal transformation) leading to $dist_{diff}(Q, P|\Theta_o) = dist(Q, P) \leq \delta$. □

V. Related Work

Relevancy to Existing Problems: As described in Section I, our 3D object retrieval problem is closely related to *registration* [13], [14], [15], [16], [17], [18] and *similarity search* [21], [22], [19], [20]. It is also related to *pattern matching* [28], [29], [30] and *object detection* [36], [37], [38] but with some differences. Pattern matching is one special case of our problem with *only one* database point cloud using another distance function. Object detection is to find a set of objects in 3D point clouds with models like deep-learning models which require some "labelled" datasets containing some objects "manually" marked as objects and could only detect objects with *known* shapes from the datasets. Our problem does not need "labelled" datasets and allows query objects with arbitrary shapes, and thus our problem is fundamentally different from object detection and is more challenging.

Relevancy to Existing Algorithms: While no existing algorithms solve our 3D object retrieval exactly, some existing algorithms (i.e., *Super4PCS* [23] and *Go-ICP* [24]) can be adapted to our problem.

Super4PCS adopts the well-known *4-point co-planar matching* scheme [13] to find the best transformation between two point clouds. It involves three steps, namely the point-pair retrieval step, the 4-point structure search step and the transformation verification step. For the sake of space, each step is detailed in our technical report [34]. Note that *Super4PCS* returns locally-optimized transformations only (not necessarily globally optimal transformation), since it follows a fundamental paradigm called RANSAC [39] that could find the best transformation in high chance (but not exactly) by repeating the above three steps many times with different random starts.

Go-ICP is the state-of-the-art algorithm to find the *globally optimal* transformation between two point clouds. In *Go-ICP*, each transformation is represented by 6 parameters and thus, the search space contains all possible values from these parameters. It searches for the best transformation by a Branch-and-bound (BnB) search strategy where each iteration of the BnB search splits a search space being considered into 8 equal-sized subspaces until the optimal transformation is found. However, *Go-ICP* is costly, with $O(8^l)$ time complexity in the worst case where l is a data-dependent parameter denoting the maximum number of iterations in the BnB search (and is about 30 on average in our experiment). Note that although the original Go-ICP has high time cost, it will become much more efficient if an initial transformation Θ close to the optimal is first given. This is because, with Θ, *Go-ICP* could find the exact optimal transformation by a fast sub-procedure.

We adapt *Super4PCS* [23] and *Go-ICP* [24] as follows.

(1) **Super4PCS-Adapt(NoIndex)**: We modify *Super4PCS* to form our exact approach as follows. Firstly, in the 4-point structure search step, we replace their heuristic error parameter by our $\triangle$ $(= \delta|Q|^{1/2})$ to determine whether two 4-point structures are "similar". Secondly, we replace their transformation verification step with our complete transformation and object retrieval steps for global optimality. Thirdly, since *Super4PCS* handles two point clouds only, we run our adapted algorithm between the query and each database point cloud. (2) **Super4PCS-Adapt(Index)**: Based on **Super4PCS-Adapt(NoIndex)**, we enhance the point-pair retrieval step with a 1-dimensional index (e.g., B+-tree) to index all pairwise distances among all points in each database point cloud. Note that in all steps of **Super4PCS-Adapt(NoIndex)**, the only step we could improve with the index is the point-pair retrieval step based on the pairwise distance search (which leads us to a 1-dimensional index). With this index, the two sets (i.e., S_1 and S_2) could be found more efficiently. (3) **GoICP-Adapt**: Since *Go-ICP* gives the *optimal* transformation between two point clouds only, we run *Go-ICP* for the query and each database point cloud. If the optimal distance is within δ, we include the corresponding database point cloud in the result set.

The non-index approach, **Super4PCS-Adapt(NoIndex)**, does not perform well (compared with our C_2O algorithm) because the 4-point search time cost is linear to the database size n [23], but C_2O takes $O(\log n + k_2)$ where k_2 is the output size. The index approach, **Super4PCS-Adapt(Index)**, does not perform well either (compared with C_2O) because the 1D index only accelerates the point-pair retrieval step, and the remaining steps for finding the matched 4-points are still time-consuming. Since the output size from the index could be as large as n, this algorithm is not efficient enough. Note that in C_2O query phase, we have a similar step to find all the relative-distance representations in the database. Our step only takes $O(\log n + k_2)$ time, where k_2 is very small ($k_2 = 40$ on average in our experiment). The rationale of improvement is that we represent and match a 4-point as a *whole* instead of handling it as two *separate* point-pairs in existing approaches. Finally, **GoICP-Adapt** does not perform well due to its high computation cost.

There are some existing algorithms solving the similarity search problem: PointNetVLAD [19], PCAN [20], feature-based descriptors [21], [40], [41], [42] and 3D shape descriptors [22], [43]. Two representative algorithms are PointNetVLAD [19] and PCAN [20] which are deep learning models trained on a number of 3D point clouds and are used for similarity search via embedding vectors as described in Section I. However, both models do not consider any rotation/translation of the given query point cloud and thus, the results from these models are not accurate. To improve their accuracy, we adapt the *training* phase of each of these two models by applying our distance measurement to label similar training samples. Moreover, methods of feature-based descriptors [21], [40], [41], [42] form *local descriptors* for similarity search that exploit neighboring geometric features of sampled key points from a point cloud, which easily give inaccurate results with noise or with different key point sampling. Methods of 3D shape descriptors [22], [43] either summarize a global descriptor from the above local descriptors [22] (so they still suffer from the same issues as feature-based descriptors), or form a rough representation of the entire point cloud using some heuristic functions [43] (so they still cannot capture the 3D shape exactly).

VI. Experiment

A. Setup

We conducted experiments on a machine with 2.66GHz CPU and 48G memory in C++.

1) Datasets: We used 3 datasets, 2 of them (namely *Object* [44] and *Indoor* [45]) from a well-structured 3D object dataset repository called *redwood* [46] (commonly used in 3D graphics [45], [7], [44]), and the other (namely *OS-MN40* [47]) from the recent 3D object retrieval challenge [48]. Dataset *Object* involves 441 3D scanned real-life objects, dataset *Indoor* has 5 3D scenes of indoor environments, and dataset *OS-MN40* involves 9,487 3D CAD objects of which 8,527 (960) objects are used for the "collection" ("query") set. Each object/scene in all datasets is in the form of a point cloud. The average *diameters* of the object/scene point clouds for the 3 datasets are 4,500mm, 1,700mm and 250,000mm, respectively, where the *diameter* of a point cloud is defined to be the diameter of the minimum bounding sphere covering this point cloud.

For each dataset, we form its database (DB) dataset as follows. For dataset *Object*, we first form a point cloud for each object of size around 100 (following [49], [50], [51]) using Quadric Edge Collapse Decimation (QECD) [52] (a seminal method implemented in MeshLab [53]), and then randomly select 400 (out of 441) objects to form a DB dataset called *Object-DB*. For dataset *Indoor*, we select 3 scenes (namely bedroom, boardroom and loft with 2.5M, 4.5M and 3M points, respectively) to form a DB dataset called *Indoor-DB*. For datasets *OS-MN40*, we form DB dataset *OS-MN40-DB* with all the 8527 objects (each object re-sampled into around 100 points using QECD) from its "collection" set.

Based on the above DB datasets, we construct additional datasets for scalability test. In dataset *Object-DB*, we create 5 datasets with the number of objects as 100, 1K, 10K, 100K and 1M, respectively. For the first dataset, we use 100 (out of 400) objects from dataset *Object-DB*. For the remaining 4 datasets, since each of them contains more than 400 objects, we pick one point cloud P from *Object-DB* and create a new object by perturbing the coordinates of each point in P with distortion values generated according to Gaussian distribution with mean equal to 0 and standard deviation equal to 0.05 times the diameter of point cloud P. The sizes of the 5 generated datasets (i.e., n) in *Object-DB* are 12.2K, 118K, 1.18M, 11.8M and 117M, respectively. In dataset *Indoor-DB*, we create 3 datasets with database size as 10K, 100K and 1M, by sampling all point clouds in dataset *Indoor-DB* using QECD. Other detailed processing (e.g., dataset re-scaling) are shown in [34].

2) Random Query Generation: We generate random queries as follows. For dataset *Object*, we randomly pick one point cloud as query from *Object-DB*. For dataset *Indoor*, we select one scene randomly from the 3 scenes in *Indoor-DB* and extract a random part from the selected scene as the query point cloud with diameter roughly equal to 1,000mm. We also vary this extraction diameter to 1,500mm, 2,000mm, 2,500mm and 3,000mm for our scalability tests. For dataset *OS-MN40*, we randomly pick one point cloud from its "query" set. Note that there is no identical object between its "query" set and its "collection" set (which is used to form its DB dataset), and thus the queries bear less similarity with the database objects, which makes the retrieval task more challenging. For each obtained query point cloud Q in all datasets, we perform a random translation and rotation on Q, and to simulate varied levels of noise, we also introduce noise levels of 10%, 20%, 30% and 40% by performing a perturbation on each coordinate value of Q following Gaussian distribution with mean equal to 0 and standard deviation equal to 0.0025 times the diameter of Q [54]. In addition, we test different types of queries for dataset *Object* as follows. (a) Non-existing and mixing types: we randomly pick one point cloud from the rest 41 objects in dataset *Object* but outside *Object-DB* for query (called the "non-existing" queries) and we also mix the queries in *Object-DB* and the *non-existing* queries together with varied proportions (details can be found in [34]). (b) Partial-matching types: we extract a random part of a query point cloud Q such that the proportion of points in each extracted part over all points in Q (called *overlap*) is 25%, 50% and 75%, respectively. For dataset *Indoor*, we form the non-existing, mixing and partial-matching types of queries similarly, and the details are shown in [34].

3) Algorithms: We include all the adapted existing algorithms (described in Section V) for comparison: **Super4PCS-Adapt(NoIndex)** [23], **Super4PCS-Adapt(Index)** [23] and **GoICP-Adapt** [24]. Our proposed algorithm is denoted by $\mathbf{C_2O}$. We also include the two state-of-the-art deep learning algorithms, denoted as **PointNetVLAD** [19] and **PCAN** [20].

Since the deep learning algorithms are designed to retrieve the top-k similar objects (or portions) in the database, we return the top-k where k is set to the number of all the expected database point clouds (or portions) for a given query and δ. Due to the architecture nature of the deep learning algorithms requiring 128 points as input, we re-sample each database/query object (or portion) with 128 points using QECD. Following [19], [20], we do additional pre-processing for dataset *Indoor*, and the details are presented in [34].

4) Factors: We vary the following factors: (1) r, (2) query point cloud size, (3) database size, (4) δ, (5) query noise percentage and (6) overlap. (1) By default, we set r to 1,200mm, 350mm and 150mm for datasets *Object*, *Indoor* and *OS-MN40*, respectively, which lead to the best (i.e., smallest) query time. (2) The default query diameter for dataset *Indoor* is 1,000mm. Unlike dataset *Indoor*, there is no need to vary query diameter for datasets *Object* or *OS-MN40* since a whole database object is specified as a query object. (3) The default dataset sizes are 118K, 10K, and 850K for datsets generated from *Object*, *Indoor* and *OS-MN40*. (4) Following [24], we set the default value of δ to be 3.2% (of the query diameter) for datasets *Object* and *Indoor*. For dataset *OS-MN40*, we set δ to be 20% (of the query diameter) since this setting gives accurate results of retrieving similar objects with the same type (as described later). (5) The default query noise percentage (as described in Section VI-A2) is 10%. (6) The default overlap is 100% (2%) for dataset *Object* (*Indoor*).

5) Measurements: We have the following measurements: (1) the index building time, (2) the index size, (3) the query time, (4) the number of query relative-distance representation candidates, (5) the number of complete transformations and (6) the precision, recall and F-measure (showing how accurate the algorithms are). All measurements are straightforward. Note that the F-measure is defined to be the harmonic mean of the precision and recall, where the precision is defined to be the proportion of the expected results retrieved by an algorithm (i.e., the true positives) over all the retrieved results, and the recall is defined to be the proportion of the retrieved expected results over all the expected results. Each measurement is reported as an average of at least 100 random query executions.

In our experiments, we obtain the expected results of each query Q based on the distance defined in Equation 3. That is, the expected results are the database point clouds with distance to Q at most δ. Later, in Section VI-B3, we show that by setting δ to the default values, our retrieval results are accurate and effective for the real-world retrieval problem.

B. Results

1) Study of Design of Our C_2O Algorithm: We conducted experiments about the design of $\mathbf{C_2O}$, i.e., choosing the best value of r (based on the smallest query time), choosing the regular tetrahedron (based on the high pruning power), choosing the strategy of selecting $q_{(1)}$ from Q (based on the smallest query time), studying the effectiveness of pruning relative-distance representations with a multi-dimensional index I_{DB}, and choosing R*-tree to implement I_{DB} (based on the smallest query time). Details could be found in [34].

2) Comparison Among All Algorithms for Efficiency: In the following, we give the experimental results for the efficiency.

Preprocessing Efficiency: We study the preprocessing time and the memory consumption of index-based algorithms (i.e., $\mathbf{C_2O}$ and **Super4PCS-Adapt(Index)**) and deep learning algorithms (i.e., **PointNetVLAD** and **PCAN**). The preprocessing time and the memory consumption of an index-based algorithm (a deep learning algorithm) refer to its index construction time (training time) and its index size (deep learning model size), respectively. As shown in Figure 8(a), the preprocessing times of all algorithms increase with the number of database objects. Though $\mathbf{C_2O}$ has slightly longer preprocessing time, it consumes much smaller memory than **Super4PCS-Adapt(Index)** that has a too bulky index size to be executed on the 1M-object database. Figure 8(b) shows that the memory consumption of index-based algorithms increases with the number of database objects, while that of deep learning algorithms remains unchanged (since their model sizes are independent of the number of objects).

Effect of Size of Database: The query time of all non-deep learning algorithms increases with the number of database objects as shown in Figure 8(c). Our $\mathbf{C_2O}$ has the shortest query time (e.g., 4.6s for 1M database objects). The fastest baseline **Super4PCS-Adapt(Index)** has 2–3 orders of magnitude longer query time than $\mathbf{C_2O}$, although it slightly improves **Super4PCS-Adapt(NoIndex)** due to its index. **GoICP-Adapt** takes the longest query time due to its costly optimal transformation for all database objects. Note that the query times of the two deep learning algorithms (i.e., **PointNetVLAD** and **PCAN**) remain nearly unchanged at around 0.6s because searching through the embedding vectors of all database objects in the K-D tree is very efficient and well-scaling to the increase of the database size. Nevertheless, in our default setting with 1K objects, our $\mathbf{C_2O}$ is still much faster (i.e., in 0.032s) than deep learning algorithms (which are later shown to be inaccurate as well). Note that the result of **PointNetVLAD** (**PCAN**) limits to the dataset with at most 10K (100K) objects, because for larger datasets, their preprocessing times are too long (e.g., more than 10^6s).

Effect of Size of Query: Figure 8(d) shows that the query times of non-deep learning algorithms increase with the size of the query (i.e., $|Q|$). When $|Q|$ increases, the parameter $\triangle$ ($= \delta|Q|^{1/2}$) also increases, resulting in increased query times for our algorithm and **Super4PCS**-related algorithms. Though $\mathbf{C_2O}$ shows a slightly larger increasing trend (since

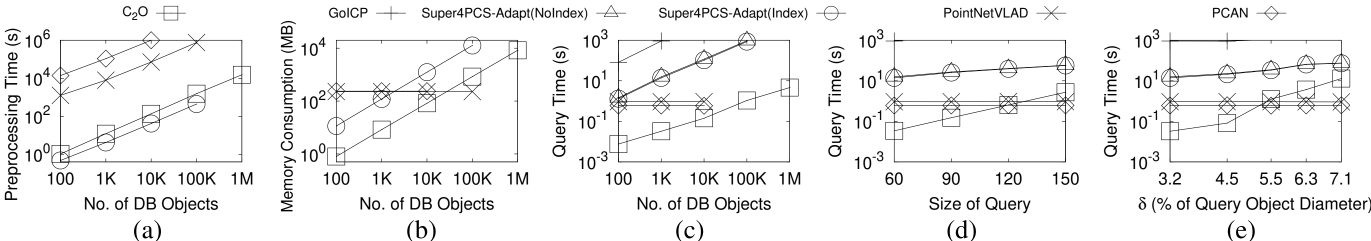


Fig. 8. Preprocessing Time (a), Memory Consumption (b), and Effect of Size of Database (c), Size of Query (d) and δ (e) on Query Time for Dataset *Object* (Similar Results for Other Datasets Presented in [34])

$\mathbf{C_2O}$ could generate a larger candidate set for a query point cloud with more points), $\mathbf{C_2O}$ still outperforms **Super4PCS-Adapt(Index)** by more than 1 order of magnitude. The query times of deep learning algorithms remain unchanged due to re-sampling to 128 points as input to their models for every query.

Effect of δ: Figure 8(e) shows that the query times of all algorithms except deep learning algorithms increase with δ. Our $\mathbf{C_2O}$ still outperforms all the non-deep learning algorithms.

Other Results regarding Efficiency: Our $\mathbf{C_2O}$ obtains similar superior efficiency as shown above for other experiments (i.e., when varying the noise percentage, when querying with non-existing and mixing types, and when testing on other datasets). In particular, on the 1M-point-database of dataset *Indoor*, the query time of our $\mathbf{C_2O}$ is within 4.1s but the exact baselines need at least 657s. For dataset *OS-MN40* (of size 850K with a larger δ ($= 20\%$)), we perform queries within 2.3s, while the exact baselines need more than 282s. See [34] for details.

3) Effectiveness of $\mathbf{C_2O}$ *for Retrieval Accuracy:* We show the effectiveness of $\mathbf{C_2O}$ in terms of retrieval accuracy. First, we observe that our $\mathbf{C_2O}$ (and any other exact algorithm) always returns query results with 100% F-measure. This verifies the ability of our $\mathbf{C_2O}$ to answer the 3D object retrieval problem exactly. However, the F-measures of deep learning algorithms are only around or less than 15% (with both precision and recall $< 25\%$), because they do not consider any rotation/translation, and thus they return irrelevant objects (i.e., incorrect results) in most cases. Moreover, the 100% F-measure result of $\mathbf{C_2O}$ is consistent in more challenging cases (e.g., high noise percentage and low overlap), while the deep learning algorithms perform even worse (e.g., around 5% F-measure when overlap $=$ 25% for dataset *Object*). All the detailed results are shown in [34]. Next, we further show our effectiveness in retrieval accuracy by case studies.

Case Studies: We conducted a number of case studies about the results returned by our $\mathbf{C_2O}$ and the deep learning algorithms. Figure 9(a) shows the query object Q (i.e., a standing sign) for dataset *Object* (note that Q could be in arbitrary position and orientation, which is simulated by a random translation and rotation in our experiments). When setting δ to our default value of this dataset (i.e., 3.2% of the query diameter), our $\mathbf{C_2O}$ returns the exact object (as shown in Figure 9(b)) as well as another standing sign that is very similar to Q (as shown in Figure 9(c)). Note that both objects have distance to Q (as defined in Equation 3 in the form of the average L_2-norm) within the default δ. However, the deep learning algorithm **PointNetVLAD** returns two objects where one of them is correct but the other is an irrelevant object (as shown in Figures 9(d) and (e)). Another case study for dataset *OS-MN40* is shown in Figure 10. In this dataset, there is no exact matching of the query object, and thus we set the default δ to 20% of the query diameter so that the similar database objects with the same type could be retrieved. In this case study, the similar benches to the query (as shown in Figure 10(a)) are returned (with two examples shown in Figures 10(b) and (c)), while most results returned by **PointNetVLAD** are irrelevant objects with other types (with two examples shown in Figures 10(d) and (e)). Deep learning algorithm **PCAN** also gives similar inaccurate results, and the details are shown in [34].

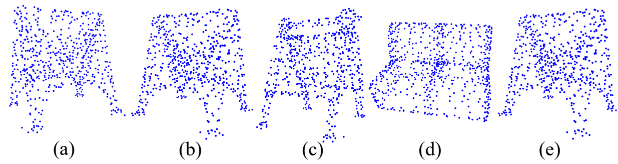


Fig. 9. Case Study of Dataset *Object* with Query Object (a) as Standing Sign ($\delta = 3.2\%$)

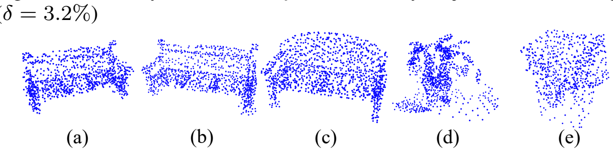


Fig. 10. Case Study of Dataset *OS-MN40* with Query Object (a) as Bench ($\delta = 20\%$)

4) Summary of Results: We verify the superior efficiency of our proposed $\mathbf{C_2O}$ algorithm for object retrieval queries, which generally outperforms the existing accurate algorithms by 1-3 orders of magnitude for various experimental configurations. In dataset *Object* with 1M objects (over 100M points), our algorithm obtains efficient and superior performance (e.g., within 5 seconds), while none of the existing accurate algorithms handle it in reasonable time (e.g., less than 1000 seconds). Moreover, our algorithm returns accurate results with 100% F-measure value, but the existing deep learning algorithms obtain at most around 15% F-measure value.

VII. Conclusion

In this paper, we studied the problem about efficient 3D object retrieval which have many applications. We propose our C_2O algorithm, which, in most of our experiments, performs up to 1–2 orders of magnitude faster than all adapted algorithms in the literature. Moreover, C_2O guarantees 100% accurate results but some adapted algorithms do not. Some possible future work could be the top-k version and the dynamic version of 3D object retrieval.

References

[1] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[2] L. Keselman, J. I. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel (r) realsense (tm) stereoscopic depth cameras," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2017, pp. 1267–1276.

[3] 2022. [Online]. Available: https://blog.google/products/maps/street-view-15-new-features/

[4] 2022. [Online]. Available: https://www.yeggi.com/

[5] S. Kim, S. Kim, and D.-E. Lee, "Sustainable application of hybrid point cloud and bim method for tracking construction progress," *Sustainability*, vol. 12, no. 10, p. 4106, 2020.

[6] A. Goparaju, K. Iyer, A. Bone, N. Hu, H. B. Henninger, A. E. Anderson, S. Durrleman, M. Jacxsens, A. Morris, I. Csecs *et al.*, "Benchmarking off-the-shelf statistical shape modeling tools in clinical applications," *Medical Image Analysis*, vol. 76, p. 102271, 2022.

[7] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5556–5565.

[8] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[9] D. Van Krevelen and R. Poelman, "Augmented reality: Technologies, applications, and limitations," *Vrije Univ. Amsterdam, Dep. Comput. Sci*, 2007.

[10] X. Chen, I. Vizzo, T. Läbe, J. Behley, and C. Stachniss, "Range image-based lidar localization for autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 5802–5808.

[11] K.-M. Cheung, W. Jun, G. Lightsey, and C. Lee, "Single-satellite real-time relative positioning for moon and mars," 2019.

[12] M. D. Samirbhai and S. Chen, "A star pattern recognition technique based on the binary pattern formed from the fft coefficients," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[13] D. Aiger, N. J. Mitra, and D. Cohen-Or, "4-points congruent sets for robust pairwise surface registration," in *ACM transactions on graphics (TOG)*, vol. 27, no. 3. Acm, 2008, p. 85.

[14] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[15] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 3212–3217.

[16] H. Li and R. Hartley, "The 3d-3d registration problem revisited," in *2007 IEEE 11th international conference on computer vision*. IEEE, 2007, pp. 1–8.

[17] J. Dong, Z. Cai, and S. Du, "Improvement of affine iterative closest point algorithm for partial registration," *IET Computer Vision*, vol. 11, no. 2, pp. 135–144, 2016.

[18] G. Elbaz, T. Avraham, and A. Fischer, "3d point cloud registration for localization using a deep neural network auto-encoder," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4631–4640.

[19] M. A. Uy and G. H. Lee, "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] W. Zhang and C. Xiao, "Pcan: 3d attention map learning using contextual information for point cloud based retrieval," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 436–12 445.

[21] B. Bustos, D. A. Keim, D. Saupe, T. Schreck, and D. V. Vranić, "Feature-based similarity search in 3d object databases," *ACM Computing Surveys (CSUR)*, vol. 37, no. 4, pp. 345–387, 2005.

[22] M. Körtgen, G.-J. Park, M. Novotni, and R. Klein, "3d shape matching with 3d shape contexts," in *The 7th central European seminar on computer graphics*, vol. 3. Citeseer, 2003, pp. 5–17.

[23] N. Mellado, D. Aiger, and N. J. Mitra, "Super 4pcs fast global pointcloud registration via smart indexing," in *Computer Graphics Forum*, vol. 33, no. 5. Wiley Online Library, 2014, pp. 205–215.

[24] J. Yang, H. Li, and Y. Jia, "Go-icp: Solving 3d registration efficiently and globally optimally," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1457–1464.

[25] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-based recognition of 3d point clouds in urban environments," in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 2154–2161.

[26] A. Pagani, C. C. Gava, Y. Cui, B. Krolla, J.-M. Hengen, and D. Stricker, "Dense 3d point cloud generation from multiple high-resolution spherical images." in *VAST*, 2011, pp. 17–24.

[27] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–606.

[28] P. J. de Rezende and D. Lee, "Point set pattern matching ind-dimensions," *Algorithmica*, vol. 13, no. 4, pp. 387–404, 1995.

[29] P. Brass, "Exact point pattern matching and the number of congruent triangles in a three-dimensional pointset," in *European Symposium on Algorithms*. Springer, 2000, pp. 112–119.

[30] L. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets, "Geometric pattern matching under euclidean motion," *Computational Geometry*, vol. 7, no. 1-2, pp. 113–124, 1997.

[31] C.-S. Chen, Y.-P. Hung, and J.-B. Cheng, "Ransac-based darces: A new approach to fast automatic registration of partially overlapping range images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 11, pp. 1229–1234, 1999.

[32] M. Mohamad, M. T. Ahmed, D. Rappaport, and M. Greenspan, "Super generalized 4pcs for 3d registration," in *2015 International Conference on 3D Vision*. IEEE, 2015, pp. 598–606.

[33] M. Mohamad, D. Rappaport, and M. Greenspan, "Generalized 4-points congruent sets for 3d registration," in *2014 2nd international conference on 3D vision*, vol. 1. IEEE, 2014, pp. 83–90.

[34] A. Authors, "On efficient 3d object retrieval (technical report)," 2023. [Online]. Available: https://github.com/anonym62371/F458812606D3F73A

[35] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *Acm Sigmod Record*, vol. 19, no. 2. Acm, 1990, pp. 322–331.

[36] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[37] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4490–4499.

[38] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 918–927.

[39] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[40] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 3384–3391.

[41] B. I. van Blokland and T. Theoharis, "An indexing scheme and descriptor for 3d object retrieval based on local shape querying," *Computers & Graphics*, vol. 92, pp. 55–66, 2020.

[42] D. Dimou and K. Moustakas, "Fast 3d scene segmentation and partial object retrieval using local geometric surface features," *Transactions on Computational Science XXXVI: Special Issue on Cyberworlds and Cybersecurity*, pp. 79–98, 2020.

[43] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3 d shape descriptors," in *Symposium on geometry processing*, vol. 6, 2003, pp. 156–164.

[44] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun, "A large dataset of object scans," *arXiv preprint arXiv:1602.02481*, 2016.

[45] J. Park, Q.-Y. Zhou, and V. Koltun, "Colored point cloud registration revisited," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 143–152.

[46] 2022. [Online]. Available: http://redwood-data.org/

[47] Y. Feng, Y. Gao, X. Zhao, Y. Guo, N. Bagewadi, N.-T. Bui, H. Dao, S. Gangisetty, R. Guan, X. Han *et al.*, "Shrec'22 track: Open-set 3d object retrieval," *Computers & Graphics*, vol. 107, pp. 231–240, 2022.
[48] 2023. [Online]. Available: https://www.shrec.net/
[49] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox, "6-dof grasping for target-driven object manipulation in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6232–6238.
[50] D. Xu, D. Anguelov, and A. Jain, "Pointfusion: Deep sensor fusion for 3d bounding box estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 244–253.
[51] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.
[52] M. Tarini, N. Pietroni, P. Cignoni, D. Panozzo, and E. Puppo, "Practical quad mesh simplification," in *Computer Graphics Forum*, vol. 29, no. 2. Wiley Online Library, 2010, pp. 407–418.
[53] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "Meshlab: an open-source mesh processing tool." in *Eurographics Italian chapter conference*, vol. 2008, 2008, pp. 129–136.
[54] Q.-Y. Zhou, J. Park, and V. Koltun, "Fast global registration," in *European Conference on Computer Vision*. Springer, 2016, pp. 766–782.
[55] J. D. Foley, A. van Dam, J. F. Hughes, and S. K. Feiner, "Spatial-partitioning representations; surface detail," *Computer Graphics: Principles and Practice*, 1990.
[56] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[57] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

APPENDIX A
TABLE OF NOTATIONS

In Table I, we summarize the commonly used notations in this paper.

APPENDIX B
EXTENDED DETAILS OF ALGORITHM C_2O

A. Extended Details of Donut Representation

In Section IV-A, we introduced the steps of forming tetrahedron, which generate a tetrahedron close to a regular tetrahedron (represented by $p_{(1)}$, $p_{(2)}$, a and b). Now, we give a lemma to formally show that the tetrahedron represented by point $p_{(1)}$, $p_{(2)}$, a and b is a regular tetrahedron.

Lemma B.1. *The tetrahedron represented by $p_{(1)}$, $p_{(2)}$, a and b is regular.*

Proof. Since m is the mid-point between $p_{(1)}$ and $p_{(2)}$, $ma \perp mp_{(1)}$ and $\|ma\| = r' = \frac{\sqrt{3}}{2}\|p_{(1)}, p_{(2)}\|$, we could derive that $\triangle p_{(1)}ap_{(2)}$ is a regular triangle. Since point b is decided by rotating ma around point m inside the perpendicular plane of $mp_{(1)}$, it holds that $mb \perp mp_{(1)}$ and $\|m, a\| = \|m, b\|$. Similarly, we know that $\triangle p_{(1)}bp_{(2)}$ is also a regular triangle. It left to show that $\|a, b\| = \|p_{(1)}, p_{(2)}\|$ such that all the edges of tetrahedron formed by point $p_{(1)}$, $p_{(2)}$, a and b are equal. Since $\theta = \arccos 1/3$, by the law of cosines, it is easy to derive that $\|a, b\| = \sqrt{2r'^2 - 2r'^2 \cos\theta} = \frac{2\sqrt{3}}{3}r' = \|p_1, p_2\|$. $\square$

B. Extended Details of Why Bound $\triangle$ Is Tight

In the following lemma, we show that the bound $\triangle = \delta|Q|^{1/2}$ introduced in Section IV-B is tight by showing that, in Lemma IV.1, it is possible that $\|q', p\| = \triangle$.

Lemma B.2. *Consider a database point cloud P. There exists a query point cloud Q such that Q satisfies the following 2 conditions. (1) $dist(Q, P) \leq \delta$ and (2) there exists a point q' in Q' such that $\|q', p\| = \triangle$, where Q' is the point cloud optimally transformed from Q (wrt P) and p is the correspondence point of q' on P.*

Proof. We construct a query point cloud Q based on a database point cloud P in $\mathcal{P}$ as follows. Let Q be $\varnothing$ initially. For each point p in P except an arbitrarily selected point, we create a point q with the same coordination as p, i.e., $q = p$, and then insert q into Q. After that, we insert into Q a point q' such that the distance between q' and the nearest point of q' in P is $\triangle$. It can be verified that $dist(Q, P) = \sqrt{\triangle^2/|Q|} = \delta$ (i.e., condition (1) is satisfied). Also, considering the point q' whose correspondence point p on p (i.e., the nearest point of q' in P), then $\|q', p\|$ is exactly $\triangle$ (i.e., condition (2) is satisfied). $\square$

C. Extended Details of Constructing $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$

In this section, we give more detailed explanation of how we construct $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$, which has been introduced in Section IV-B1, Section IV-B2 and Section IV-B3, respectively.

We give the following lemma based on the concepts of the sphere bounding zone, which can help to construct $S_{(2)}$. The

Notation	Description
$\mathcal{P}$	A database
n	The size of a database
Q	The query point cloud
P	A point cloud in the database
Θ	A rigid transformation
$T_\Theta(p)$	The transformed point of point p wrt Θ
$T_\Theta(P)$	The transformed point cloud of P wrt Θ
$corr(q, P)$	The correspondence point of point q on P
$\|q, p\|$	The Euclidean distance between point q and point p
$dist(Q, P)$	The distance between Q and P in different coordinate systems
$rd(p_{(1)}\|p_{(2)}, p_{(3)}, p_{(4)})$	The relative-distance representation of point $p_{(1)}$ wrt $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$
$ball(p, r)$	The r-sized ball of p
$ball\text{-}NN(p, r\|P)$	The nearest neighbor of the r-sized ball of p in P
$donut(p, r, \mathbf{n})$	The r-sized donut of p with its normal $\mathbf{n}$
$donut\text{-}NN(p, r, \mathbf{n}\|P)$	The nearest neighbor of the r-sized donut of p with its normal $\mathbf{n}$ in P
$sbz(p, [r_1, r_2])$	The sphere boundary zone of p with interval $[r_1, r_2]$
$dbz(D, [r_1, r_2])$	The donut boundary zone of D with interval $[r_1, r_2]$

TABLE I

COMMONLY USED NOTATIONS

idea is that we would like to find a small subset of Q, denoted by $S_{(2)}$, which must contain the desired $q_{(2)}$ having $p_{(2)}$ as the correspondence point on P. Since $p_{(2)}$ is the nearest neighbor of the r-sized ball of $p_{(1)}$ in P (and is very close to the r-sized ball of $p_{(1)}$ in most cases), based on the two major principles, it is guaranteed that our desired $q_{(2)}$ is also close to the r-sized ball of $q_{(1)}$ or the distance from our desired $q_{(2)}$ to $q_{(1)}$ is close to the distance from $p_{(2)}$ to $p_{(1)}$.

Lemma B.3. *Consider a query point cloud Q and a database point cloud P. Let $q_{(1)}$ be a point in Q whose correspondence point on P is $p_{(1)}$. Let $p_{(2)}, p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of $p_{(1)}$, respectively. Let $S_{(2)}$ be the set of all points from Q in $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$ where $q' = ball\text{-}NN(q_{(1)}, r + 2\triangle|Q)$. If $dist(Q, P) \le \delta$, then there exists a point $q_{(2)}$ in $S_{(2)}$ such that $p_{(2)}$ is the correspondence point of $q_{(2)}$ on P.*

Proof sketch. Since $dist(Q, P) \le \delta$, by Lemma IV.1, $q_{(2)}$ exists in Q. Then, we need to show $r - 2\triangle \le \|q_{(1)}, q_{(2)}\| \le \|q_{(1)}, q'\| + 4\triangle$. If not, then it must be contradicted to the definition of q' and $p_{(2)}$ and conditions $\|q_{(1)}, p_{(1)}\| \le \triangle$ and $\|q_{(2)}, p_{(2)}\| \le \triangle$. □

The full proof can be found in Section C-D.

According to the above lemma, given $q_{(1)}$ (whose correspondence point on P is $p_{(1)}$), we can find a set $S_{(2)}$ of *candidate* points in Q for $q_{(2)}$, such that $S_{(2)}$ must contain $q_{(2)}$ which has its correspondence point on P as $p_{(2)}$, where $p_{(2)}$ is the second owner of the relative-distance representation of $p_{(1)}$ and can be listed out.

Next, suppose that we are given $q_{(1)}$ and $q_{(2)}$ (whose correspondence points on P are $p_{(1)}$ and $p_{(2)}$, respectively). Following the same steps introduced in Section IV-A, we find the locus of a point, denoted by D, with distance to both $q_{(1)}$ and $q_{(2)}$ equal to $\|q_{(1)}, q_{(2)}\|$ (i.e., $D = donut(m, r', \mathbf{n})$, where m is the mid-point between $q_{(1)}$ and $q_{(2)}$, $r' = \frac{\sqrt{3}}{2}\|q_{(1)}, q_{(2)}\|$ and $\mathbf{n}$ is the vector from $q_{(2)}$ to $q_{(1)}$). Note that in Section IV-A, for the *database* point cloud P, the third owner selected for the relative-distance representation of $p_{(1)}$ (i.e., $p_{(3)}$) is the nearest point in P to a donut (constructed based on the points in P). For this *query* point cloud Q, we also have a similar mechanism for finding the third point in Q. Based on the property that the distance from each query point to its correspondence point can be bounded (by Lemma IV.1), we use our defined concept in Section IV-B2 called donut boundary zone to capture this property. We show the following lemma for constructing $S_{(3)}$ with the donut D derived from $q_{(1)}$ and $q_{(2)}$.

Lemma B.4. *Consider a query point cloud Q and a database point cloud P. Let $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of point $p_{(1)}$ in P, respectively. Let $q_{(1)}$ and $q_{(2)}$ be the points in Q whose correspondence point on P are $p_{(1)}$ and $p_{(2)}$, respectively. Let m be the mid-point between $q_{(1)}$ and $q_{(2)}$, $r' = \frac{\sqrt{3}}{2}\|q_{(1)}, q_{(2)}\|$, and $\mathbf{n}$ be the vector from $q_{(2)}$ to $q_{(1)}$. Let $D = donut(m, r', \mathbf{n})$ and $q' = donut\text{-}NN(m, r', \mathbf{n}|Q)$. Let $S_{(3)}$ be the set of all points from Q in $dbz(D, [0, dist(q', D) + 4\triangle])$. If $dist(Q, P) \le \delta$, then there exists a point $q_{(3)}$ in $S_{(3)}$ such that $p_{(3)}$ is the correspondence point of $q_{(3)}$ on P.*

Proof. Since $dist(Q, P) \le \delta$, by Lemma IV.1, $q_{(3)}$ exists in Q. Since q' is the nearest point in Q to D, no points in Q is inside $dbz(D, [0, dist(q', D)))$ (which does not include the points with distance to D exactly $dist(q', D)$), thus we only need to show that $dist(q_{(3)}, D) \le dist(q', D) + 4\triangle$. Suppose $dist(q_{(3)}, D) > dist(q', D) + 4\triangle$, there exists point $p' \ne p_{(3)}$ in P, such that p' has smaller distance to D_p (the donut we build for P) than $p_{(3)}$, which contradicts that $p_{(3)}$ is the nearest point to D_p. □

According to the above lemma, given $q_{(1)}$ and $q_{(2)}$ (whose correspondence points on P are $p_{(1)}$ and $p_{(2)}$, respectively), we can also find a set $S_{(3)}$ of *candidate* points in Q for $q_{(3)}$ inside region $dbz(D, [0, dist(q', D) + 4\triangle])$ such that $S_{(3)}$ must contain $q_{(3)}$ which has its correspondence point on P as $p_{(3)}$, where $p_{(3)}$ is the third owner of the relative-distance representation of $p_{(1)}$ and can be listed out.

Finally, if we are given $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ (whose correspondence points on P are $p_{(1)}$, $p_{(2)}$ and $p_{(3)}$, respectively), we still follow the steps in Section IV-A to find point a (a point on donut D that is nearest to $q_{(3)}$) and point b (the point at the position closest to a in the anti-clockwise direction from a on D in the view point from $q_{(1)}$ to $q_{(2)}$). *Suppose* that we follow the original "next" step in Section IV-A which is to find the point in Q nearest to point b as the fourth point in the relative-distance representation. Similarly, due to the property that the distance from each query point to its correspondence point can be bounded (by Lemma IV.1), instead of following the original step *exactly*, we need to find some points in Q near to b as candidates of the fourth point $q_{(4)}$ in the relative-distance representation of $q_{(1)}$. Specifically, these points could be found in a space represented by a sphere centered at b. The following lemma shows the definition of this sphere and we could find $S_{(4)}$ containing these points.

Lemma B.5. *Consider a query point cloud Q and a database point cloud P. Let $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$ be the second, third, fourth owners of the relative-distance representation of point $p_{(1)}$ in P, respectively. Let $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ be the points in Q whose correspondence point on P are $p_{(1)}$, $p_{(2)}$ and $p_{(3)}$, respectively. Let m be the mid-point between $q_{(1)}$ and $q_{(2)}$, $r' = \frac{\sqrt{3}}{2}\|q_{(1)}, q_{(2)}\|$, and $\mathbf{n}$ be the vector from $q_{(2)}$ to $q_{(1)}$. Let $D = donut(m, r', \mathbf{n})$. Let a be a point on D that is nearest to $q_{(3)}$. Let b be the point at the position nearest to a in the anti-clockwise direction from a on D in the view point from $q_{(1)}$ to $q_{(2)}$ (such that $q_{(1)}$, $q_{(2)}$, a and b form a regular tetrahedron). Let q' be the nearest neighbor of b in Q. Let $S_{(4)}$ be the set of all points from Q in $sbz(b, [0, \|b, q'\| + 4\triangle])$. If $dist(Q, P) \leq \delta$, then there exists a point $q_{(4)}$ in $S_{(4)}$ such that $p_{(4)}$ is the correspondence point of $q_{(4)}$ on P.*

Proof. Since $dist(Q, P) \leq \delta$, by Lemma IV.1, $q_{(4)}$ also exists in Q. Applying the similar idea in the proof of Lemma B.4, we show $\|b, q_{(4)}\| \leq \|b, q\| + 4\triangle$. If $\|b, q_{(4)}\| > \|b, q\| + 4\triangle$, then there exists a point in P which has smaller distance to point b than $p_{(4)}$, which contradicts that $p_{(4)}$ is the nearest neighbor of b. $\square$

Based on the above lemma, given $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ (whose correspondence points on P are $p_{(1)}$, $p_{(2)}$ and $p_{(3)}$, respectively), we can find a set $S_{(4)}$ of *candidate* points in Q for $q_{(4)}$ inside region $sbz(b, [0, \|b, q'\| + 4\triangle])$, such that $S_{(4)}$ must contain $q_{(4)}$ which has its correspondence point on P as $p_{(4)}$, where $p_{(4)}$ is the fourth owner of the relative-distance representation of $p_{(1)}$ and can be listed out.

D. Extended Details of Rotation and Translation Invariant Property of Candidate Set Generation

In this section, we give the following lemma to show that the output candidate set generated by the steps of constructing the candidate set of relative-distance representations introduced in Section IV-B4 has the translation-invariant and rotation-invariant properties.

Lemma B.6. *Consider a query point cloud Q. Let $q_{(1)}$ be a point in Q. Let $\Theta = (R, t)$ be a transformation consisting of a rotation matrix R and a translation vector t. Consider another query point cloud Q' such that $Q' = T_\Theta(Q)$. Let $q'_{(1)}$ be a point in Q'. Let $\mathcal{C}_{q_{(1)}}$ ($\mathcal{C}'_{q_{(1)}}$) be the output of the steps to find the set of candidate relative-distance representations in Q (Q') with the starting point $q_{(1)}$ ($q'_{(1)}$). If $q'_{(1)} = T_\Theta(q_{(1)})$, then $\mathcal{C}_{q_{(1)}} = \mathcal{C}'_{q_{(1)}}$.*

Proof sketch. We need to show (a) $\forall R \in \mathcal{C}_{q_{(1)}}$, $\exists R' \in \mathcal{C}'_{q_{(1)}}$ such that $R = R'$ and (b) the other direction of (a). For (a), consider $q'_{(i)} = T_\Theta(q_{(i)})$ for all $i \in [1, 4]$ where $q_{(i)}$ ($q'_{(i)}$ are the owners of R (R'). Then, we need to show (1) $R = R'$ and (2) $R' \in \mathcal{C}'_{q_{(1)}}$. (1) obviously holds for a rigid transformation. (2) can be proved by showing that the 3 candidate sets in Q' with $q'_{(1)}$ contains the same points as the 3 candidate sets in Q with $q_{(1)}$. (b) can be proved by swapping $\mathcal{C}'_{q_{(1)}}$ with $\mathcal{C}_{q_{(1)}}$ and replacing Θ with Θ^{-1}. $\square$

The full proof can be found in Section C-E.

E. Extended Query Complexity Analysis

In this section, we analyze the time complexity of our relative-distance representation candidate generation step in the query phase (introduced Section IV-C2) and the complexity of the number of generated candidates wrt the size of query.

Recall the steps of constructing the candidate set in Section IV-B4 starting from a point $q_{(1)}$. Given a query point cloud Q and the starting point $q_{(1)}$, we first find set $S_{(2)}$ containing points in Q inside the sphere bounding zone $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$, which takes $O(\log|Q| + |S_{(2)}|)$ time. Then, for each point $q_{(2)}$ in $S_{(2)}$, we find set $S_{(3)}$ containing points in Q inside the donut bounding zone $dbz(D, [0, dist(q', D) + 4\triangle])$, which takes $O(\log|Q| + |S_{(3)}|)$ time. Next, for each point $q_{(3)}$ in $S_{(3)}$, we find set $S_{(4)}$ containing points in Q inside the sphere bounding zone $sbz(b, [0\ \|b, q''\| + 4\triangle])$, which takes $O(\log|Q| + |S_{(4)}|)$ time. We denote M to be the average number of points in the generated sets (i.e., $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$) among all the steps. Thus, the time complexity of constructing the candidate set from $q_{(1)}$ is approximately $O(M^3 \log|Q|)$, and the total number of generated candidate representations in Q is approximately $O(M^3)$.

It is clear that each of the generated sets (i.e., $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$) contains query points inside a "thin" zone whose depth is $O(\triangle)$. But, in the worst case, M could be as large as $O(|Q|)$. Correspondingly, the total number of generated candidate representations could be as large as $O(|Q|^3)$ in the worst case. However, in practice, the bound $\triangle$ ($= \delta|Q|^{1/2}$) is small. Thus, M is small and the number of query candidates is also small. In a typical experimental setting (e.g., with the database size 1M and the query size 100), M is about 15 on average and the number of query candidates is about 400 on average. Figure 13(b) (which is introduced later in the additional experimental results) shows the number of query candidates with the varied size of query. It can be seen that, in practice, the growth of the number of query candidates is much smaller than the worst-case cubic complexity wrt the query size.

Next, since we find the query point with the smallest candidate set by an extensive search, the overall time complexity of candidate generation (i.e., Step 1 of the query phase in Section IV-C2) is $O(|Q|M^3 \log |Q|)$ in the worst case. Then, for each candidate, we find a set of database relative-distance representations in the index I_{DB} by a window query. Let R be the expected number of representations in this set. With efficient R*-tree index to implement I_{DB}, the expected time to complete the window query is $O(\log n + R)$, where n is the database size. At this point, the time complexity for performing all the window queries is thus $O(M^3(\log n + R))$.

Finally, for each window query result (totally $O(M^3 R)$ results in expectation), we perform a complete transformation using Go-ICP [24]. Although the practical time for Go-ICP with a initial transformation close to optimal is fast, the worst time complexity could still be $O(8^l)$, where l is a data dependent parameter in Go-ICP. Therefore, the overall time complexity of our query phase is $O(M^3(|Q| \log |Q| + \log n + 8^l R))$.

F. Extended Details of a Heuristic Acceleration Strategy

Although the two steps of our C_2O query phase introduced in Section IV-C2 are efficient (using index I_{DB}), we want to speed up our process with a strategy called "Relevancy Filtering" (RF).

The major idea of the RF strategy is given as follows. Note that after we choose *only one* point $q_{(1)}$ from Q in Step 1, we obtain the 2-tuple candidate set $\mathcal{C}$ of $q_{(1)}$ in Step 2b where $\mathcal{C}$ contains a number of 2-tuples each in the form of (R_Q, R_P). We could use the RF strategy to prune some of the 2-tuples in $\mathcal{C}$ with the following two steps to be introduced.

The first step in the RF strategy is performed just after Step 1 (and just before Step 2). Specifically, we generate a set $\mathcal{H}$ of h additional points in Q, namely $q'_{(1),1}, q'_{(1),2}, \ldots, q'_{(1),h}$, such that these h additional points are the h nearest points of $q_{(1)}$ in Q. For each point in $\mathcal{H}$, says $q'_{(1),j}$ where $j \in [1, h]$, we could also obtain a 2-tuple candidate set $\mathcal{C}'_j$ of $q'_{(1),j}$ (instead of $q_{(1)}$) which has the same procedure as Step 2b.

The second step in the RF strategy is performed just after Step 2b. Before we describe this second step, we first introduce a concept of "closeness" and a concept of "relevancy". Given two points $p_{(1)}$ and $p'_{(1)}$ in P and two points $q_{(1)}$ and $q'_{(1)}$ in Q, we say that pair $(p_{(1)}, p'_{(1)})$ is $\triangle$*-close* to pair $(q_{(1)}, q'_{(1)})$ if the following condition holds.

$$\|q_{(1)}, q'_{(1)}\| - 2\triangle \leq \|p_{(1)}, p'_{(1)}\| \leq \|q_{(1)}, q'_{(1)}\| + 2\triangle \quad (5)$$

In other words, when $\triangle$ is small, the distance between $p_{(1)}$ and $p'_{(1)}$ is "roughly" equal to the distance between $q_{(1)}$ and $q'_{(1)}$. If the distance between $q_{(1)}$ and $q'_{(1)}$ is small, the distance between $p_{(1)}$ and $p'_{(1)}$ is small. This $\triangle$*-closeness* relationship is the key idea for our second step which is to prune some 2-tuples in the candidate set $\mathcal{C}$. Consider a 2-tuple (R_Q, R_P) in $\mathcal{C}$. When $\triangle$ is small and the distance between $q_{(1)}$ and $q'_{(1)}$ is small, if pair $(p_{(1)}, p'_{(1)})$ is not $\triangle$-close to pair $(q_{(1)}, q'_{(1)})$, we know that the distance between $p_{(1)}$ and $p'_{(1)}$ is large. We could prune this 2-tuple since we expect that the distance between two query points (i.e., $q_{(1)}$ and $q'_{(1)}$) is "roughly" equal to the distance between two "matched" database points (i.e., $p_{(1)}$ and $p'_{(1)}$).

Given a 2-tuple (R_Q, R_P) in the 2-tuple candidate set $\mathcal{C}$ of a point $q_{(1)}$ in Q and a 2-tuple (R'_Q, R'_P) in the 2-tuple candidate set of another point $q'_{(1)}$ in Q, we say that R_P is *relevant* to R'_P wrt $(q_{(1)}, q'_{(1)})$ if (1) the ID of R_P is equal to the ID of R'_P and (2) for the first owner of R_P, namely $p_{(1)}$, and the first owner of R'_P, namely $p'_{(1)}$, pair $(p_{(1)}, p'_{(1)})$ is $\triangle$-close to pair $(q_{(1)}, q'_{(1)})$. In other words, we know that (1) the two relative-distance representations (i.e., R_P and R'_P) are in the same database point cloud and (2) the distance between the first owner of R_P and the first owner of R'_P is small if the distance between $q_{(1)}$ and $q'_{(1)}$ is small (and $\triangle$ is small).

In the above definition, we define the relevancy of a representation to another representation. Next, we overload the concept of relevancy to define the relevancy of a representation to a 2-tuple candidate set. Given (1) a 2-tuple (R_Q, R_P) in the 2-tuple candidate set $\mathcal{C}$ of a point $q_{(1)}$ in Q and (2) the 2-tuple candidate set $\mathcal{C}'$ of another point $q'_{(1)}$ in Q, we say that R_P is *relevant* to $\mathcal{C}'$ wrt $(q_{(1)}, q'_{(1)})$ if there exists a 2-tuple (R'_Q, R'_P) in $\mathcal{C}'$ such that R_P is *relevant* to R'_P wrt $(q_{(1)}, q'_{(1)})$.

With the concept of "closeness" and the concept of "relevancy", we are ready to describe this second step. This second step is to remove each 2-tuple in the form of (R_Q, R_P) from $\mathcal{C}$ if this 2-tuple does *not* satisfy the *complete relevancy* condition. Given (R_Q, R_P) in the candidate set $\mathcal{C}$ (of $q_{(1)}$), (R_Q, R_P) is said to satisfy the *complete relevancy* condition if for each candidate set $\mathcal{C}'_j$ (of point $q'_{(1),j}$) where $j \in [1, h]$, R_P is relevant to $\mathcal{C}'_j$ wrt $(q_{(1)}, q'_{(1),j})$. The reason is given as follows. Suppose that (R_Q, R_P) does not satisfy the complete relevancy condition. As long as we can find one point $q'_{(1),j}$ in $\mathcal{H}$ such that R_P is not relevant to the candidate set $\mathcal{C}'_j$ of $q'_{(1),j}$ wrt $(q_{(1)}, q'_{(1),j})$, for each 2-tuple $(R'_{Q,j}, R'_{P,j})$ in $\mathcal{C}'_j$, R_P is not relevant to $R'_{P,j}$ wrt $(q_{(1)}, q'_{(1),j})$, and thus, we cannot obtain the "$\triangle$-closeness" relationship between $(p_{(1)}, p'_{(1),j})$ and $(q_{(1)}, q'_{(1),j})$ where $p_{(1)}$ ($p'_{(1),j}$) is the first owner of R_P ($R'_{P,j}$). It is worth mentioning that for the database point cloud P with the same ID as R_P, if $dist(Q, P) \leq \delta$, then for each $j \in [1, h]$, there exists a 2-tuple $(R'_{Q,j}, R'_{P,j})$ in $\mathcal{C}'_j$ such that (1) the ID of $R'_{P,j}$ is also equal to the ID of P and (2) we can obtain the "$\triangle$-closeness" relationship between $(p_{(1)}, p'_{(1),j})$ and $(q_{(1)}, q'_{(1),j})$ where $p_{(1)}$ ($p'_{(1),j}$) is the first owner of R_P ($R'_{P,j}$). Therefore, we know that "$dist(Q, P) \leq \delta$" does not hold for P, and thus, (R_Q, R_P) can be pruned.

In the following, we present a lemma to show the correctness of using the RF strategy. Specifically, let $\mathcal{C}_{RF}$ be the resulting candidate set of the RF strategy (i.e., $\mathcal{C}_{RF}$ contains all the 2-tuples in $\mathcal{C}$ that are not pruned). For a database point cloud P, if $dist(Q, P) \leq \delta$, $\mathcal{C}_{RF}$ will still contain a tuple, says (R_Q, R_P), such that each owner of R_P is the correspondence point of an owner of R_Q on P.

Lemma B.7. *Consider a query point cloud Q and a database* *point cloud P. Let $q_{(1)}$ be a point in Q whose correspondence point on P is $p_{(1)}$. Let $p_{(2)}, p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of $p_{(1)}$, respectively. Let $\mathcal{C}_{RF}$ be the resulting candidate set after performing the steps of the RF strategy with the starting point $q_{(1)}$. If $dist(Q, P) \leq \delta$, then there exists a 2-tuple (R_Q, R_P) in $\mathcal{C}_{RF}$, such that $R_P = rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ and $p_{(i)}$ is the correspondence point of $q_{(i)}$ on P for $i \in [2, 4]$, where $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$ are the second, third and fourth owners of R_Q, respectively.*

Proof. Let $\mathcal{C}$ be the 2-tuple candidate set before we perform the second step of the RF strategy (i.e., before we prune some 2-tuples with the RF strategy). By the proof of Theorem IV.1 (introduced in Section C-C), we know that there exists a 2-tuple (R_Q, R_P) in $\mathcal{C}$, such that $R_P = rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ and $p_{(i)}$ is the correspondence point of $q_{(i)}$ on P for $i \in [2, 4]$, where $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$ are the second, third and fourth owners of R_Q, respectively. Now, it is only left to show that (R_Q, R_P) will not be pruned in the RF strategy. That is, (R_Q, R_P) satisfies the *complete relevancy* condition.

Consider an arbitrary point in $\mathcal{H}$, says $q'_{(1),j}$, which is used to obtain the 2-tuple candidate set $\mathcal{C}'_j$ (note that we do not perform any pruning on $\mathcal{C}'_j$). Since $q'_{(1),j}$ is an arbitrary point in $\mathcal{H}$, we then just need to show that R_P is relevant to $\mathcal{C}'_j$ wrt $(q_{(1)}, q'_{(1),j})$. Let $p'_{(1),j}$ be the correspondence point of $q'_{(1),j}$ on P. Let $p'_{(2),j}, p'_{(3),j}$ and $p'_{(4),j}$ be the second, third and fourth owners of the relative-distance representation of $p'_{(1),j}$, respectively. Since $dist(Q, P) \leq \delta$, identically, we know that there also exists a 2-tuple $(R'_{Q,j}, R'_{P,j})$ in $\mathcal{C}'_j$, such that $R'_{P,j} = rd(p'_{(1),j}|p'_{(2),j}, p'_{(3),j}, p'_{(4),j})$ and $p'_{(i),j}$ is the correspondence point of $q'_{(i),j}$ on P for $i \in [2, 4]$, where $q'_{(2),j}$, $q'_{(3),j}$ and $q'_{(4),j}$ are the second, third and fourth owners of $R'_{Q,j}$, respectively. According to the definition of the relevancy of R_P to $\mathcal{C}'_j$ wrt $(q_{(1)}, q'_{(1),j})$, we can complete the proof by showing that R_P is relevant to $R'_{P,j}$ wrt $(q_{(1)}, q'_{(1),j})$.

The first condition is that R_P and $R'_{P,j}$ have the same ID, which is obvious since they both equal to the ID of P. The second condition is that pair $(p_{(1)}, p'_{(1)})$ is $\triangle$-close to pair $(q_{(1)}, q'_{(1)})$ (i.e., Equation 5 is satisfied). By Claim C.1 (introduced in Section C-B), since $p_{(1)}$ $(p'_{(1)})$ is the correspondence point of $q_{(1)}$ $(q'_{(1)})$ on P, Equation 5 is satisfied. □

APPENDIX C
REMAINING PROOF

A. Proof of Lemma IV.1

Proof. Applying Equation 3 where $Q' = T_{\Theta_o}(Q)$, we have $dist(Q, P) = [\frac{1}{|Q|}\sum_{q \in Q'} \|q', corr(q', P)\|^2]^{1/2} \leq \delta$, and it is trivially derived that $\sum_{q \in Q'} \|q', corr(q', P)\|^2 \leq \delta^2|Q|$. Therefore,

$$\begin{aligned} \|q', p\| &= (\|q', p\|^2)^{1/2} \\ &\leq [\sum_{q \in Q'} \|q', corr(q', P)\|^2]^{1/2} \\ &\leq (\delta^2|Q|)^{1/2} = \delta|Q|^{1/2} \end{aligned}$$

□

B. Proof of Lemma IV.2

Proof. According to Lemma IV.1, since $dist(Q, P) \leq \delta$, there exists $q_{(i)} \in Q$ for all $1 \leq i \leq 4$, such that $\|q_{(i)}, p_{(i)}\| \leq \triangle$. By Lemma B.3, $S_{(2)}$ must contain $q_{(2)}$. By Lemma B.4, $S_{(3)}$ must contain $q_{(3)}$. By Lemma B.5, $S_{(4)}$ must contain $q_{(4)}$. Therefore, the candidate relative-distance representation $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)})$ will be included in the result set $\mathcal{C}_{q_{(1)}}$. □

C. Proof of Theorem IV.1

Proof. To show $\mathcal{R} = \mathcal{R}^*$, (that is, our algorithm returns the correct result set), we just need to show that there is neither false positive (FP) nor false negative (FN) in our result. Firstly, it is obvious to see that our results do not contain any false positive, because in Step 2c, we only insert the database point clouds within the δ distance threshold of query Q into the result set. Next, to show there is no FN, we show that for any database point cloud $P \in \mathcal{R}^*$, $P \in \mathcal{R}$.

By Lemma IV.2, since $dist(Q, P) \leq \delta$, the candidate set $\mathcal{C}_{q(1)}$ must contain a "desired" relative-distance representation $R_Q = rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)})$ that corresponds with an indexed relative-distance representation $R_P = rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ in P. When we perform the window query for R_Q with query range $2\triangle$, R_P must also be one of the results by the definition of relative-distance representation (i.e., Equation 4). This is because, $\forall i \neq j \in [1, 4]$, it holds that $\|q_{(i)}, q_{(j)}\| - 2\triangle \leq \|p_{(i)}, p_{(j)}\| \leq \|q_{(i)}, q_{(j)}\| + 2\triangle$ (by Lemma IV.1). As a result, the 2-tuple (R_Q, R_P) (where R_P is associated with the ID of P) exists in $\mathcal{C}$. Next, in Step 2c, since we have included (R_Q, R_P) in $\mathcal{C}$, we will finally perform a complete transformation Θ_o between Q and P. Since we use Go-ICP [24] which ensures that Θ_o is the optimal transformation such that $dist_{diff}(Q, P|\Theta_o)$ is minimized, we will obtain the result that $dist_{diff}(Q, P|\Theta_o) = dist(Q, P) \leq \delta$. This indicates that the ID of P will be inserted into the returned set $\mathcal{R}$.

□

D. Proof of Lemma B.3

Proof. Firstly, we have the following claim, which can be easily derived from Lemma IV.1 and will be useful in our proof.

Claim C.1. *Consider a query point cloud Q and a database point cloud P. Let q and q' be two points in Q whose correspondence point in P are respectively p and p'. If $dist(Q, P) \leq \delta$, then we have*

$$\|p, p'\| - 2\triangle \leq \|q, q'\| \leq \|p, p'\| + 2\triangle$$

$$\|q, q'\| - 2\triangle \leq \|p, p'\| \leq \|q, q'\| + 2\triangle$$

Also, according to Lemma IV.1, since $dist(Q, P) \leq \delta$, there exists $q_{(2)} \in Q$, such that $\|q_{(1)}, p_{(1)}\| \leq \triangle$ and $\|q_{(2)}, p_{(2)}\| \leq \triangle$. Now, we show that $q_{(2)}$ is in $S_{(2)}$, which is the set of all points from Q in $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$ where $q' = ball\text{-}NN(q_{(1)}, r + 2\triangle|Q)$. By definition, we need to show that $r - 2\triangle \leq \|q_{(1)}, q_{(2)}\| \leq \|q_{(1)}, q'\| + 4\triangle$.

Since $q' = ball\text{-}NN(q_{(1)}, r + 2\triangle|Q)$, we have $\|q_{(1)}, q'\| \geq r + 2\triangle \geq r - 2\triangle$. Thus, we show that none of the following cases is possible: $\|q_{(1)}, q_{(2)}\| < r - 2\triangle$ or $\|q_{(1)}, q_{(2)}\| > \|q_{(1)}, q'\| + 4\triangle$.

(1) Since $p_{(2)} = ball\text{-}NN(p_{(1)}, r|P)$, we have $\|p_{(1)}, p_{(2)}\| \geq r$. Therefore, by Claim C.1, $\|q_{(1)}, q_{(2)}\| \geq \|p_{(1)}, p_{(2)}\| - 2\triangle \geq r - 2\triangle$, indicating that the first inequality cannot hold.

(2) Assuming the third inequality holds, since $\|q_{(1)}, q_{(2)}\| \leq \|p_{(1)}, p_{(2)}\| + 2\triangle$ (by Claim C.1), we have $\|p_{(1)}, p_{(2)}\| > \|q_{(1)}, q'\| + 2\triangle$. Let p' be the correspondence point to q' in P. Again by Claim C.1, we have $\|p_{(1)}, p'\| \in [\|q_{(1)}, q'\| - 2\triangle, \|q_{(1)}, q'\| + 2\triangle]$, and thus we must have $\|p_{(1)}, p'\| < \|p_{(1)}, p_{(2)}\|$. Moreover, since $\|q_{(1)}, q'\| \geq r + 2\triangle$, the lower bound of $\|p_{(1)}, p'\|$ is r, indicating that point p' is closer to the r-sized ball centered at $p_{(1)}$. This leads to a contradiction with $p_{(2)} = ball\text{-}NN(p_{(1)}, r|P)$.

Therefore, we have $q_{(2)} \in sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$. Since $q_{(2)}$ is a point of Q, by definition, we have $q_{(2)} \in S_{(2)}$.

□

E. Proof of Lemma B.6

Proof. We show that $\mathcal{C}_{q_{(1)}} = \mathcal{C}'_{q'_{(1)}}$ by the following two claims: (1) for each $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) \in \mathcal{C}_{q_{(1)}}$, there exists a relative-distance representation $rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)}) \in \mathcal{C}'_{q'_{(1)}}$ such that $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$ and (2) for each $rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)}) \in \mathcal{C}'_{q'_{(1)}}$, there exists a relative-distance representation $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) \in \mathcal{C}_{q_{(1)}}$ such that $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$.

For the first claim, consider that $q'_{(2)} = T_\Theta(q_{(2)})$, $q'_{(3)} = T_\Theta(q_{(3)})$ and $q'_{(4)} = T_\Theta(q_{(4)})$. Next, we need to show that $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$ and $rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)}) \in \mathcal{C}'_{q'_{(1)}}$.

Firstly, since for each $i \in [1, 4]$, $q'_{(i)}$ is the transformed point of $q_{(i)}$ by the same transformation Θ, it is obvious that the distance between each pair-wise distance among the four points remains unchanged. Therefore, by the definition of relative-distance representation (i.e., Equation 4), it is easy to get $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$.

Secondly, we perform the steps to find the set of candidate relative-distance representations with the same Q, r and $\triangle$ starting from $q_{(1)}$ and $q'_{(1)}$, respectively, as two *instances* denoted by $\mathcal{I}$ and $\mathcal{I}'$, respectively. We continue our proof by showing that we obtain the sets of same points in $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ for $\mathcal{I}$ and $\mathcal{I}'$, and in the following we say that $q \in Q$ and $q' \in Q'$ are the same points if $q' = T_\Theta(q)$.

(1) Let q' (q'') be the $ball\text{-}NN$ operation result we obtain (i.e., $ball\text{-}NN(q_{(1)}, r+2\triangle|Q)$) for $\mathcal{I}$ ($\mathcal{I}'$). Since Q' is obtained by transforming every point in Q with the same transformation Θ, we know that the distance between every other point to $q_{(1)}$ remains unchanged. Therefore, the $ball\text{-}NN$ operation result also remains unchanged (i.e., $q'' = T_\Theta(q')$). And then, we can obtain the same set of points inside $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$ for $\mathcal{I}$ and $\mathcal{I}'$. Further, let $S'_{(2)}$ be the set we obtain in Step (i) for $\mathcal{I}'$. By the above result, $S'_{(2)}$ must contain $q'_{(2)}$ which equals to $T_\Theta(q_{(2)})$.

(2) In Step (i)(1), we focus on the case of $q_{(2)}$ ($q'_{(2)}$) for $\mathcal{I}$ ($\mathcal{I}'$). Let m', $\mathbf{n}'$, r'' be the result of the mid-point between $q'_{(1)}$ and $q'_{(2)}$, the vector from $q'_{(2)}$ to $q'_{(1)}$ and $\frac{\sqrt{3}}{2}\|q'_{(1)}, q'_{(2)}\|$, respectively, for $\mathcal{I}'$. Then, it is also obvious that $m' = T_\Theta(m)$, $\mathbf{n}' = T_\Theta(\mathbf{n})$ and $r'' = r'$. As such, we also obtain a donut D' for $\mathcal{I}'$ which contains exactly the same points as D. Again, due to rigid transformation, the distance between every point in Q to D remains unchanged. Thus, we get the same resultant point for the $donut\text{-}NN$ operation and thus we can obtain the same set of points inside $dbz(D, [0, dist(q', D) + 4\triangle])$ for $\mathcal{I}$ and $\mathcal{I}'$. Let $S'_{(3)}$ be the set we obtain in Step (i)(1) for $\mathcal{I}'$. By the above result, $S'_{(3)}$ must contain $q'_{(3)}$ which equals to $T_\Theta(q_{(3)})$.

(3) In Step (i)(1)(I), we focus on the case of $q_{(3)}$ ($q'_{(3)}$) for $\mathcal{I}$ ($\mathcal{I}'$). Let a' and b' be the two virtual points for $\mathcal{I}'$. Since again all the points remain their distance to D, then we have $a' = T_\Theta(a)$ and $b' = T_\Theta(b)$. Thus, we can obtain the same set of points inside $sbz(b, [0\ \|b, q''\| + 4\triangle])$ for $\mathcal{I}$ and $\mathcal{I}'$. Let $S'_{(4)}$ be the set we obtain in Step (i)(1)(I) for $\mathcal{I}'$. By the above result, $S'_{(4)}$ must contain $q'_{(4)}$ which equals to $T_\Theta(q_{(4)})$. Finally, we will obtain a relative-distance representation $R = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$ and include it in the result set $\mathcal{C}'_{q'_{(1)}}$.

Till now, we have proved the first claim. Consider the reverse transformation of Θ (i.e., Θ^{-1}). Then, we have $Q = T_{\Theta^{-1}}(Q')$ and $q_{(1)} = T_{\Theta^{-1}}(q'_{(1)})$. Thus, we can directly prove the second claim by swapping Q with Q', swapping $q_{(1)}$ with $q'_{(1)}$ and substituting Θ with Θ^{-1}.

□

APPENDIX D
ADDITIONAL EXPERIMENTAL SETUP

We give the additional experimental setup in detail.

A. Datasets

For dataset *Object*, all the 441 objects fall in 44 categories (e.g., chairs, cars, standing signs, etc.). The diameter of each point cloud in dataset *Object* is 3,000–6,000mm. For dataset *Indoor*, the 5 scenes are namely bedroom, boardroom, loft, apartment and lobby. The diameter of each point cloud in dataset *Indoor* is approximately 250,000mm. For dataset *OS-MN40*, all the objects in both the "collection" and "query" sets fall in 32 categories (e.g., benches, beds, laptops, etc.). The diameter of each point cloud in dataset *OS-MN40* is 1,400–1,700mm.

For those datasets, we formed their DB datasets (i.e., *Object-DB*, *Indoor-DB* and *OS-MN40-DB*) serving for the database purpose, as we introduced in Section VI-A1. We also form the "outside-DB" datasets which will be used in the query generation (to be described later in detail) for some of the datasets. Specifically, in dataset *Object*, we chose the remaining 41 objects (outside *Object-DB*) to form a new dataset called *Object-OutsideDB*, In dataset *Object*, we chose the remaining 2 scenes (outside *Indoor-DB*, namely, apartment and lobby) to form a new dataset called *Indoor-OutsideDB*.

Specially, we re-scale the three additional datasets of *Indoor-DB* (for scalability test) to their diameters around 10,000mm, 30,000mm and 80,000mm, respectively, such that the average point-pairwise distance of each dataset is similar to that of (the original) dataset *Indoor-DB*.

B. Random Query Generation

We generated random queries with 5 types. The first-type queries are generated according to objects from the database, which correspond to the normal queries as we introduced in Section VI-A2. The second-type queries are generated according to objects outside the database, which correspond to the non-existing queries as we introduced in Section VI-A2. Specifically, for experiments related to *Object* (*Indoor*), we randomly pick one point cloud from dataset *Object-OutsideDB* (*Indoor-OutsideDB*) as the query point cloud. Then, we perform random transformation and noise perturbation similar to the first type of queries. The third-, fourth- and fifth-type queries are generated according to different distributions of the first- and second-type queries, which correspond to the mixing queries as we introduced in Section VI-A2. The third-type (fourth-type) queries are generated from 80% (20%) first-type queries and 20% (80%) second-type queries. The fifth-type queries are generated from 50% first-type queries and 50% second-type queries. By default, we use the first-type queries for experiments. We use other types of queries for other specified experiments. To test partial matching between query and database point clouds (as also introduced in Section VI-A2), we also extract the random parts containing the target number of points from a selected scene such that the overlap is varied from 2% to 10%.

C. Algorithms

Following the pre-processing steps of the deep learning algorithms [19], [20], we split each database point cloud of dataset *Indoor* into a number of small point clouds using a voxel grid [55] of grid size equal to our default query diameter (to be introduced later) where each small point cloud (containing all the points within a cell) represents a portion of a database point cloud. For some experiments where the query diameter is varied for dataset *Indoor*, we set the grid size to different values accordingly to form different pre-processed *Indoor* datasets. Moreover, we follow [19], [20] to mark a retrieved portion as correct if its center has its Euclidean distance to the center of the expected portion within the diameter of the query point cloud (that is, it is very close to the expected portion).

APPENDIX E
ADDITIONAL EXPERIMENTAL RESULTS

We show our additional experimental results in the following parts. In Section E-A, we show the detailed results of studying the design of our C_2O algorithm. In Section E-B, we show the additional results for dataset *Object*. In Section E-C, we show all the experimental results for dataset *Indoor*. In Section E-D, we show the results for other datasets.

A. Detailed Results of Studying the Design of Our C_2O Algorithm

Study of Parameter in our C_2O framework: We studied the major parameter in our C_2O framework, namely r (i.e, the side length of a tetrahedron). For the default setting of dataset *Object*, we build our $\mathbf{C_2O}$ index with varied radius r ranging from 500mm to 1,500mm. We take the average values to report experimental results.

In this experiment, after we set r to a value, we obtain an index based on a number of tetrahedra constructed in our $\mathbf{C_2O}$ index. Based on each index, we could issue queries described before and measure the query time. As illustrated in Figure 11(a), when r increases, the query time decreases first, reaches the minimum query time when r reaches around 1,200mm and increases after that. The reason is that smaller tetrahedra (with smaller r values) tend to trigger more (expensive) complete transformations in the database, which is consistent with existing observations [32] (since it is very likely that a small given tetrahedron is "similar" to a lot of small tetrahedra due to the *micro-view* (or too detailed view) from this given tetrahedron, resulting in a non-distinguishable tetrahedron). However, when r is very large, the cost of spatial operations (e.g. queries finding *ball-NN*) becomes higher and thus, the query time is larger. According to this, we set r = 1,200mm leading to the best-performing $\mathbf{C_2O}$ for the *Object* databases, and for dataset *Indoor*, we set r = 350mm following the similar trends in our experiments.

Study of 4-point Structure Compared with 3-point Structure: In this paper, we use the concept of regular tetrahedron to construct our donut representation. This concept involves a structure consisting of 4 points, which could lead to significantly better differentiating power than using 3-point structure [13]. We verified this conclusion by showing that the 4-point structure outperforms 3-point structure dramatically in both the number of complete transformations and query time.

We compared **Super4PCS-Adapt(Index)** (the best existing algorithm using 4-point structure [13]) and our $\mathbf{C_2O}$ algorithm with the adapted state-of-the-art 3-point structure approach [31] (denoted by **3PCS-Adapt(Index)**). Specifically, (1) we randomly select 3 points from Q to form a query 3-point structure Γ, (2) we find a set G of candidate 3-point structures in all database point clouds such that each candidate has similar structure as Γ within error parameter $\triangle$ (which is similar to the modification of **Super4PCS-Adapt(Index)**), (3) for each candidate 3-point structure Λ in G, we perform the coarse transformation based on Λ and Γ, and then perform complete transformation and object retrieval steps based on the coarse transformation result. Notably, we also apply the one-dimensional index built for **Super4PCS-Adapt(Index)** to accelerate the above Step (2), which also includes a same point-pair retrieval step in the middle.

As shown in Figure 11(b), the number of complete transformations of the existing 4-point algorithm (i.e.,

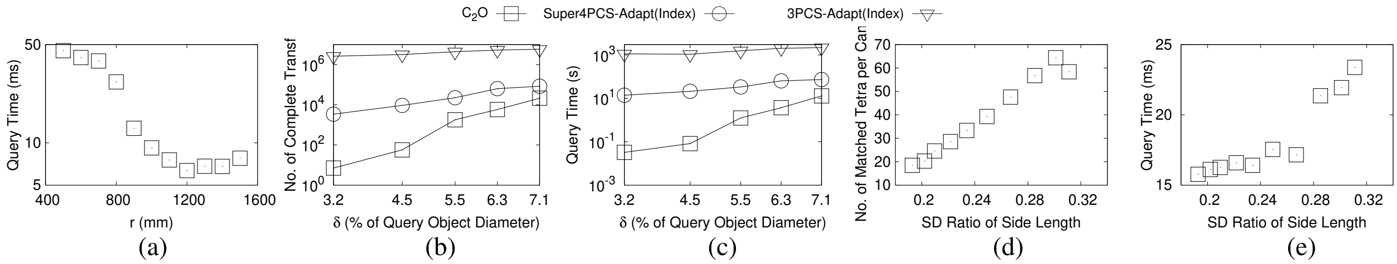


Fig. 11. Effect of Indexed Tetrahedra Size, 4-Point Structure and Regularity of $\mathbf{C_2O}$ for Dataset *Object*

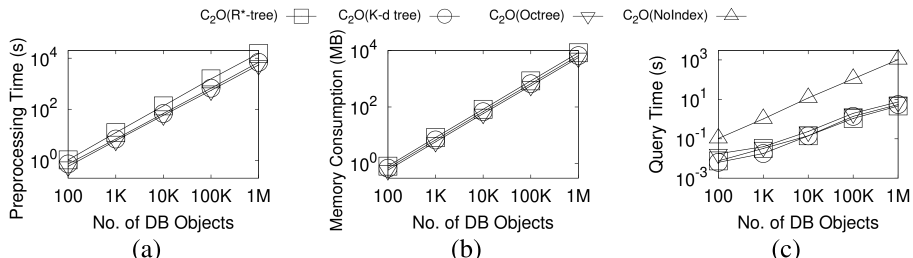


Fig. 12. Effect of Index Types for Dataset *Object*

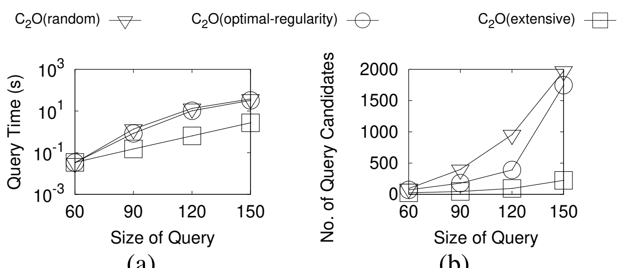


Fig. 13. Effect of Strategies of Selecting $q_{(1)}$ for Dataset *Object*

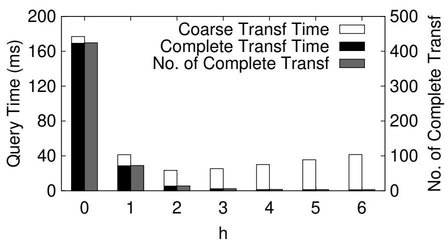


Fig. 14. Effect of the *RF* Strategy

Super4PCS-Adapt(Index)) continue to outperform that of **3PCS-Adapt(Index)** by around 2 orders of magnitude as parameter δ increases. As a result, as shown in Figure 11(c), **Super4PCS-Adapt(Index)** also needs around 2 orders of magnitude less query time than **3PCS-Adapt(Index)**. It is worth mentioning that our $\mathbf{C_2O}$ algorithm even significantly outperforms **Super4PCS-Adapt(Index)** using the donut representation.

Study of Regularity of Tetrahedron: Next, we study why the regularity of a tetrahedron, the major principle used in our C_2O algorithm, is used. We also used the same experimental setup as the above experiment to build our $\mathbf{C_2O}$ index. In this experiment, the regularity of a tetrahedron could be described by the SD (Standard Deviation) ratio of side length which is defined to be the SD over the 6 side lengths divided by the average side length. A more regular tetrahedron tends to have similar side lengths, resulting in a smaller SD. Here, adopting the (relative) ratio (i.e., the SD divided by the average side length) instead of the absolute SD value is to study the regularity (which is relative in nature). We measure the number of database tetrahedra matched for each query tetrahedron candidate for each indexed tetrahedron.

Figure 11(d) shows that, in general, the number of database tetrahedra matched for each query tetrahedron candidate increases when the SD ratio increases. This means that less regularity (larger SD ratio) leads to more matched tetrahedra to be verified, leading to a larger query time as shown in Figure 11(e). This could justify our strategy of using more regular tetrahedra (where the SD ratio is smaller).

Study of Index Types: Since our C_2O algorithm can allow different types of multi-dimensional index to implement I_{DB}, we test three types of commonly used multi-dimensional index, namely R*-tree [35], k-d tree [56] and octree [57]. The three variants are indicated by $\mathbf{C_2O}$**(R*-tree)**, $\mathbf{C_2O}$**(K-d tree)** and $\mathbf{C_2O}$**(Octree)**, respectively. Note that for the octree variant, we use the first three dimensions in each 6-dimensional relative-distance representation for indexing, since the octree structure will have poor query performance when generalizing to higher dimensions. We also include a baseline here, indicated by $\mathbf{C_2O}$**(NoIndex)**, for our C_2O algorithm without building any multi-dimension index for efficient search (and thus the search is implemented by linear scanning all the relative-distance representations in the database).

As shown in Figure 12, the three variants have similar performance for all the related measurements (i.e., index building time, index size and query time), and they all outperform the baseline $\mathbf{C_2O}$**(NoIndex)** in query performance. Particularly, $\mathbf{C_2O}$**(Octree)** has the (slightly) best index building time and (slightly) smallest index size, while $\mathbf{C_2O}$**(R*-tree)** has the (slightly) slowest index building time and (slightly) largest index size. However, $\mathbf{C_2O}$**(R*-tree)** has the best query efficiency among the three variants, due to its superior performance of organizing points in the 6-dimension space. Compared with the baseline $\mathbf{C_2O}$**(NoIndex)**, using the R*-tree index leads to two orders of magnitude improvement on query time when the database scales to 1M objects. Due to its superior query efficiency, we use the R*-tree index as the default implementation of our C_2O algorithm.

Study of Strategies of Selecting $q_{(1)}$: In the query phase of C_2O, we use a strategy of selecting $q_{(1)}$ which is to extensively search for the point $q_{(1)}$ (among all points in Q) such that the number of the generated candidate set is the smallest. To verify the effectiveness of this strategy (which is denoted as $\mathbf{C_2O}$**(extensive)**), we compared it with another two strategies. The first strategy (which is denoted as $\mathbf{C_2O}$**(random)**) is to randomly pick a point in Q as $q_{(1)}$. The second strategy is based on the following heuristic. We form a *regularity measurement* of a point q in Q based on the r*-surrounding set* of q in Q where r is the radius parameter to construct our C_2O index. The r-surrounding set of q in Q is defined to be a set containing q itself and 3 other points in Q where these 3 points are the nearest to the surface of the sphere centered at q with radius r. Our regularity measurement of a point q in Q is defined to be the SD ratio of the tetrahedron formed by the r-surrounding set of q in Q. Intuitively, if the regularity measurement of q in Q is larger, then the r-surrounding set of q in Q can form a tetrahedron which is has a more regular shape. This could lead to the result of stronger pruning power. Therefore, the second strategy is to find the point in Q (and assign it to $q_{(1)}$) such that the regularity measurement is optimal (i.e., the smallest) among all points in Q by computing the regularity measurement of each point in Q. We denote the second stragy as $\mathbf{C_2O}$**(optimal-regularity)**.

As shown in Figure 13(a), when the size of query increases, all the three strategies need longer time to run the query. Except in the case of the smallest query size, the extensive search strategies achieves much better query efficiency than the other two strategies, which indicates better scalability. Recall that the number of candidates of query relative-distance representations is an important factor influencing the query time. As shown in Figure 13(b), the number of candidates for $\mathbf{C_2O}$**(extensive)** is much smaller than the other two strategies (since it always find the smallest candidate set), and thus $\mathbf{C_2O}$**(extensive)** is the most efficient.

Effect of Relevancy Filtering (RF) Strategy: In Section B-F, we propose the *RF* strategy to reduce the number of complete transformations by selecting h additional nearby query points for pruning. We now show how much query time this strategy improves and what value h to select by varying h from 0 (our algorithm without RF) to 6.

As illustrated in Figure 14, the *RF* strategy improves the overall query time, including both the coarse transformation time and the complete transformation time, significantly. Although the coarse transformation time (white bar) increases with h (since we perform more coarse transformation operations), the complete transformation time (black bar) decreases dramatically since the number of complete transformations

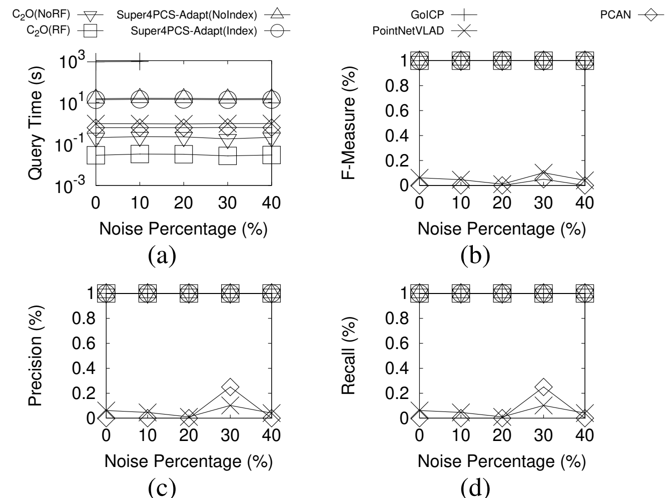


Fig. 15. Effect of Noise Percentage for Dataset *Object*

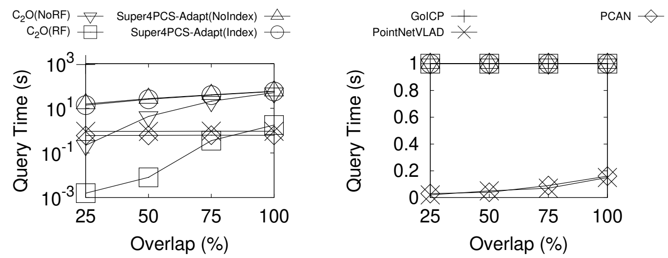


Fig. 16. Effect of Overlap for Dataset *Object*

(grey bar) decreases by 1–2 orders of magnitudes. However, when $h > 2$, the coarse transformation time begins to dominate the total query time, causing the total query time to increase slightly. We thus fix h to be 2 in the rest of the experiments for the best performance. For the remaining experimental results, and we use $\mathbf{C_2O}$**(RF)** to denote our algorithm with the RF strategy. We also include the original algorithm (denoted by $\mathbf{C_2O}$**(NoRF)**) to fully show the effectiveness of the RF strategy.

B. Additional Results on Dataset Object

1) Effect of Noise Percentage on Dataset Object*:* Figure 15(a) shows that the noise percentage does not affect the query times of all algorithms. In particular, $\mathbf{C_2O}$**(RF)** gives the smallest query time. Figure 15(b) shows that the F-measure of all algorithms except deep learning algorithms is 100% but that of deep learning algorithms is only around or less than 15%. Figure 15(c) and (d) show that both the precision and recall (which are the breakdown measurements of the F-measure) of all algorithms except deep learning algorithms are also 100% but deep learning algorithms obtain only less than 25% for both precision and recall.

Effect of Overlap: We vary the overlap between query and database point clouds from 25% to 100%. Similarly, $\mathbf{C_2O}$**(RF)** is the most efficient among all exact algorithms and still performs accurately. In lower-overlap cases (e.g., overlap = 25%), the F-measure of deep learning algorithms is even smaller (e.g., around 5%), because it is more difficult for deep learning algorithms to capture the shape of query objects when

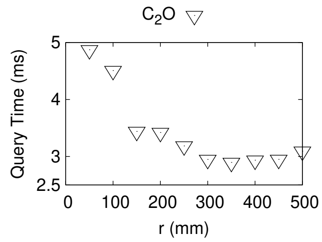
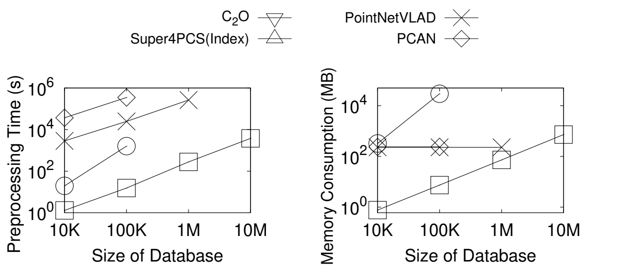
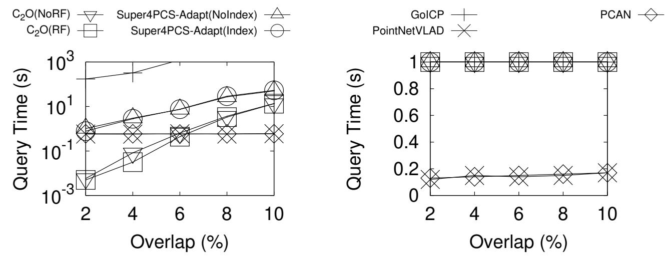


Fig. 17. Effect of Indexed Tetrahedra Size for Dataset *Indoor*



(a) (b)

Fig. 18. Preprocessing Time and Memory Consumption for Dataset *Indoor*



(a) (b)

Fig. 19. Effect of Overlap for Dataset *Indoor*

the overlap is low. Nevertheless, $\mathbf{C_2O(RF)}$ always has 100% F-measure.

2) Results of Different Types of Queries on Dataset Object*:* In this part, we demonstrate the comparison among all algorithms of dataset *Object* for different types of queries as described in Section VI-A2. Specifically, the results for the second-type, third-type, fourth-type and fifth-type queries are reported in Figure 20, 21, 22 and 23, respectively.

The result we obtain for each of the other query types is similar to that the first type of queries (as shown in Figure 8 and 15). For the types of queries which contain more query objects outside the database (i.e., the second-type queries 100% of which are outside the database, and the fourth-type queries 80% of which are outside the database), we observe that all the non-deep learning algorithms use slightly more time to execute the queries (as shown in Figure 20 and 22, respectively) than the other types, but our proposed algorithms still have the shortest query times among all the non-deep learning algorithms for all the query types. Specifically, in the default setting, when the type of queries is switched from the first-type to the second-type, the query time of our $\mathbf{C_2O(RF)}$ algorithm increases from 0.032s to 0.04s, while the query time of the fastest existing non-deep learning algorithm (i.e., **Super4PCS-Adapt(Index)**) increases from 13.9s to 22s.

C. Experimental Results on Dataset Indoor

In this part, we report the detailed experimental results on dataset *Indoor*.

First, we show the query time of our algorithm $\mathbf{C_2O}$ with the effect of r in Figure 17. We set r ranging from 50mm to 500mm for dataset *Indoor*. We observe the similar trend as for dataset *Object* (as shown in Figure 11(a)). Specifically, the query time first drops when r increases from 50mm to 350mm and then increases slightly when r is larger than 350mm. Therefore, we set $r = 350$mm for all the experiments on dataset *Indoor* since it leads to the smallest query time.

Next, we also show the comparison results of preprocessing time and memory consumption for all the algorithms that need preprocessing for dataset *Indoor*. The results show that, for dataset *Indoor*, our preprocessing step is significantly faster than that of the existing index-based algorithm and all the deep learning algorithms (as shown in Figure 18(a)). Moreover, as shown in Figure 18(b), the memory consumption of our algorithm $\mathbf{C_2O}$ is also much more efficient than that of the index-based algorithm **Super4PCS-Adapt(Index)** which reaches about 20GB even when the size of database is 100K. Note that the memory consumption of our algorithm $\mathbf{C_2O}$ is only about 800MB for the largest size of database with over 10M points.

Then, we present results for varying the overlap in Figure 19, which also show similar results as in dataset *Object*. Note that the overlap between the default query (with diameter 1,000mm) and the database scene is only 2%, which verifies our capability of addressing a partial matching problem where the query is only a small part inside the large database scene. In this setting, since the query diameter is still in a reasonable range, we ensure that the matched results are still the meaningful part of the database scene (instead of noise).

We also run queries on dataset *Indoor* with all types of queries for all the factors we study. The results are reported in Figure 24, 25, 26, 27 and 28 for the first-type, second-type, third-type, fourth-type and fifth-type queries, respectively. All the results are similar to those on dataset *Object*.

D. Experimental Results on Other Datasets

We also report the experimental results on other datasets. In dataset *OS-MN40*, we run random queries for the database *OS-MN40-DB*. The size of this database this 850K (with 8,527 objects each of size around 100). Parameter δ is set to 20% for this dataset. The average query time of our $\mathbf{C_2O}$ is 2.274s with 100% F-measure. The query times of exact baselines **GoICP-Adapt**, **Super4PCS-Adapt(NoIndex)** and **Super4PCS-Adapt(Index)** are 1643s, 298.3s and 282.1s, respectively. The query times of deep learning algorithms **PointNetVLAD** and **PCAN** are 0.634s and 0.793s, respectively, but they only obtain F-measure 24% and 19%, respectively.

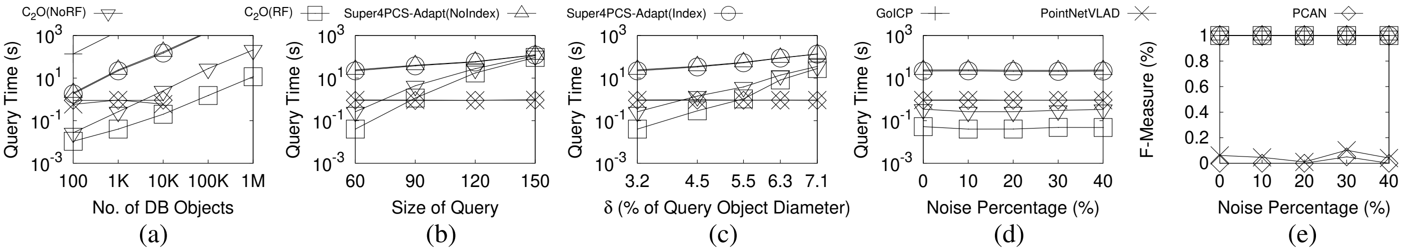


Fig. 20. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Second-type Queries

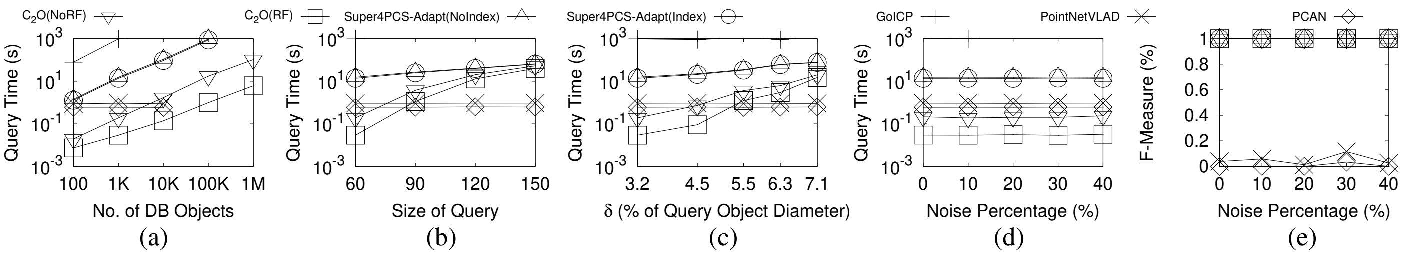


Fig. 21. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Third-type Queries

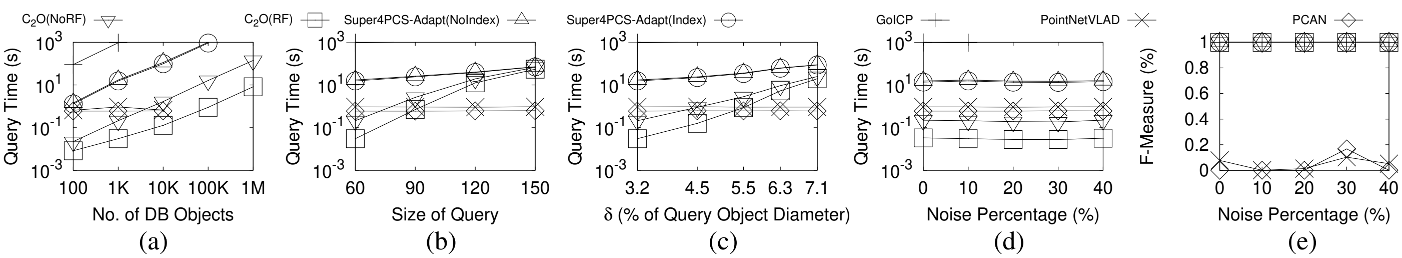


Fig. 22. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Fourth-type Queries

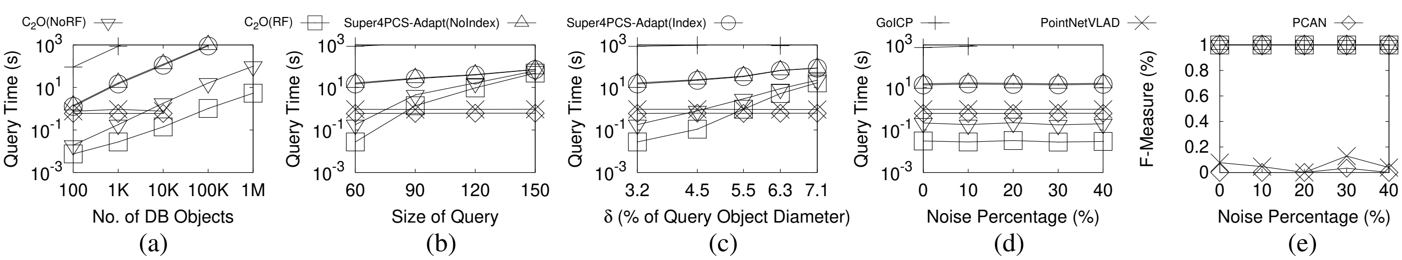


Fig. 23. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Fifth-type Queries

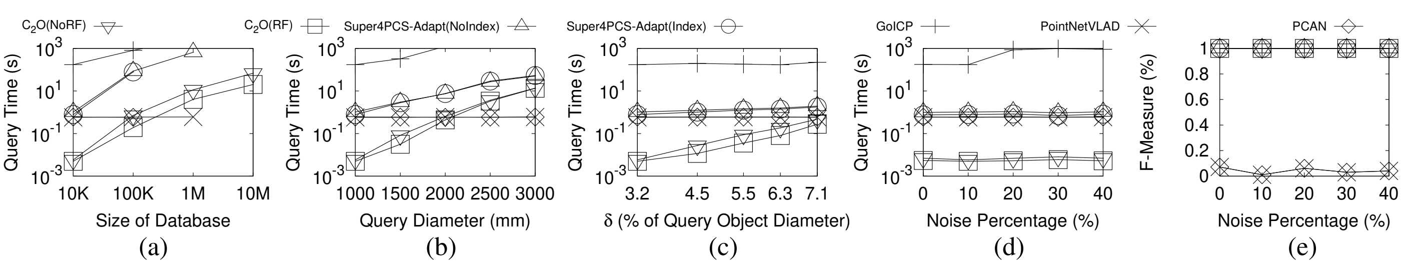


Fig. 24. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the First-type Queries

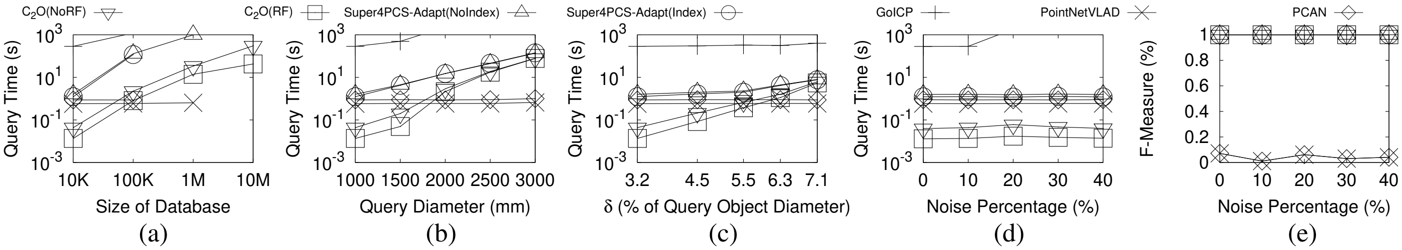


Fig. 25. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Second-type Queries

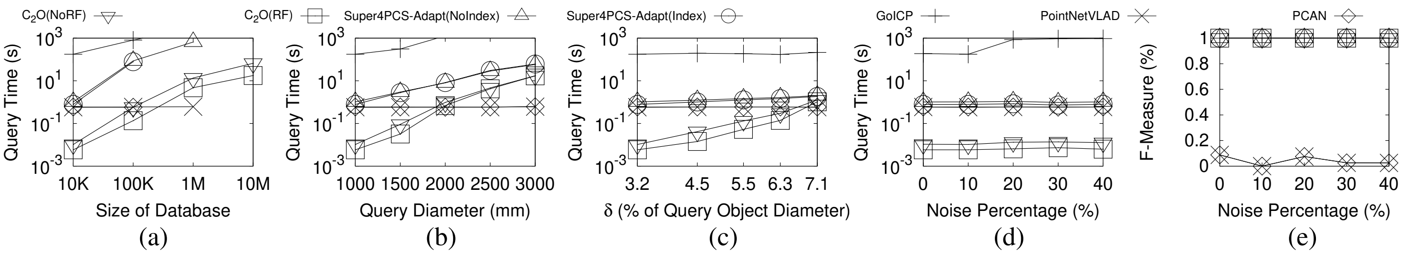


Fig. 26. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Third-type Queries

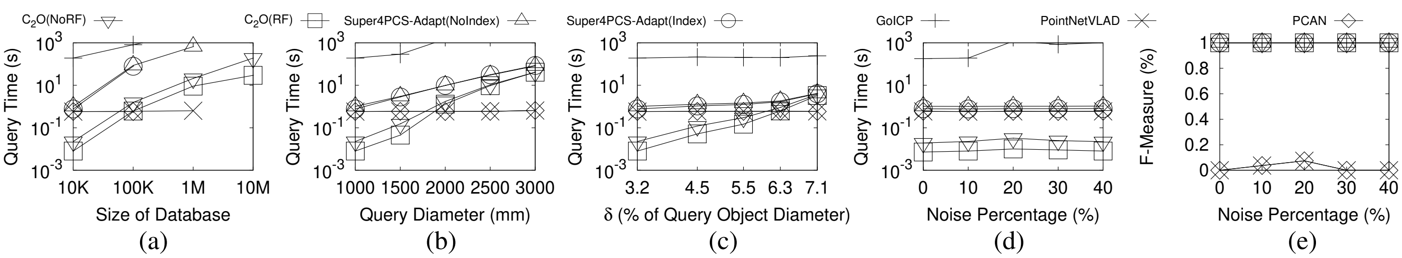


Fig. 27. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Fourth-type Queries

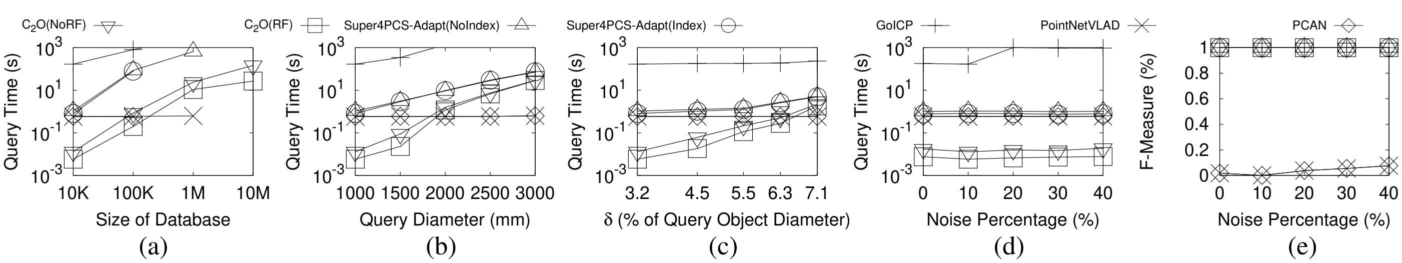


Fig. 28. Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Fifth-type Queries